Algorithmes et protocoles pour la gestion des données et des calculs dans les environnements distribués et hétérogènes

Emmanuel Jeannot

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Algorithmes et protocoles pour la gestion des données et des calculs dans les environnements distribués et hétérogènes

MÉMOIRE

présenté et soutenu publiquement le 26 octobre 2007

pour l'obtention de l'

Habilitation à Diriger des Recherches de l'université Henri Poincaré

(spécialité informatique)

par

Emmanuel Jeannot

Composition du jury

Rapporteurs :	Michel Daydé, Professeur à l'Enseith, Toulouse Brigitte Plateau, Professeur à l'ENSIMAG, Grenoble Thierry Priol, Directeur de recherche à l'INRIA, Rennes
Examinateurs :	Jack Dongarra, Professeur à l'Université du Tennessee, Knoxville Jens Gustedt, Directeur de recherche à l'INRIA, Nancy Dominique Mery, Professeur à l'UHP, Nancy
Invité :	Franck Cappello, Directeur de recherche à l'INRIA, Orsay

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Remerciements

La vie et le travail de chercheur n'est pas une aventure solitaire. Bien souvent, les résultats et les contributions scientifiques sont le fruit d'échanges, de collaborations et de discussions. Les travaux présentés ici n'échappent pas à cette règle et c'est donc bien volontiers que je remercie ceux qui, d'une manière ou d'une autre, ont rendu possible cet aboutissement qu'est une habilitation à diriger des recherches.

Mes premiers remerciements vont à ceux qui, à leur manière, m'ont appris ce merveilleux métier de la recherche. Michel Cosnard est le premier d'entre eux et même si depuis la fin de ma thèse notre collaboration scientifique est devenue plus qu'épisodique, les échanges que nous avons eus restent des souvenirs extrêmement forts autant d'un point de vue professionnel qu'humain. Frédéric Desprez et Jean Roman m'ont encadré pendant mon post-doc au LaBRI. Cette expérience très enrichissante m'a permis d'aborder une nouvelle thématique (les grilles) qui est au cœur de ce mémoire. Depuis que je suis au LORIA, j'ai la chance de travailler aux côtés de Jens Gustedt avec qui j'ai créé notre équipe de recherche et qui m'a ouvert les yeux sur le rôle fondamental de l'expérience en informatique. Je dois ma première expérience de la recherche aux USA à Jack Dongarra qui m'a accueilli à Knoxville pendant l'été 1994 et plus récemment en 2006. À chaque fois ce fut une épopée très marquante. Parmi les seniors que j'ai eu l'occasion de côtoyer, Luc Bougé et Yves Robert sont probablement ceux qui ont eu le plus d'influence sur moi de par leurs enseignements et leurs encouragements. Enfin, j'ai eu l'occasion d'effectuer ma première expérience de recherche au LRI avec Cécile Germain. Je la remercie chaleureusement de m'avoir donné ce premier aperçu.

À mon tour, j'ai eu la chance d'encadrer des étudiants et des ingénieurs venant d'horizons très divers. J'espère leur avoir donné le goût de la recherche et je les remercie pour m'avoir aidé dans la réalisation de ces travaux. Il s'agit, en particulier et dans l'ordre chronologique de Laurent Bobelin, Bertrand Cirou, Yves Caniou, Nicolas Padoy, Frédéric Wagner, Louis-Claude Canon, Vandy Berten, Xavier Delaruelle, Zhiao Shi, Olivier Dubuisson et Eloi Du Bois.

Pendant ces longues années, j'ai eu l'honneur et le plaisir de côtoyer et de travailler avec de nombreux collègues : Frédéric Suter, Martin Quinson, Luiz Angello Steffenel, Joël Gossens, Christian Perez, Stéphane Vialle, Asim YarKhan, Keith Seymour, Georges Bosilca, Julien Langou, Denis Trystram, Jean-Marie Moureaux, Johanne Cohen, Franck Cappello, Raymond Namyst, Stéphane Genaud, Eddy Caron, Frédéric Vivien, Jean-Louis Pazat, Denis Caromel et bien d'autres.

Je n'oublie pas non plus que pendant cinq ans, j'ai été maître de conférences. Les conditions du département ont été un réel avantage pour mener à bien cette HDR. Je veux donc ici remercier mes ex-collègues pour leur accueil et la bonne ambiance qu'ils ont toujours su entretenir : Jean-Marie Moureaux, Dominique Richier, Didier Fradet, Thierry Divoux, David Brie, Hervé Ortéga et tous les autres.

Je tiens ensuite à remercier les rapporteurs, Michel Daydé, Brigitte Plateau et Thierry Priol, pour avoir eu l'obligeance d'accepter d'évaluer ce manuscrit malgré un emploi du temps que je sais très chargé. Tous mes remerciements à Franck Cappello, Jack Dongarra, Jens Gustedt et Dominique Mery pour avoir accepté d'être membres de mon jury.

Je ne voudrais pas non plus oublier le personnel administratif et en particulier les assistantes et les secrétaires sans qui notre travail serait beaucoup plus compliqué, en particulier Madeleine Mann, Josiane Reffort et Céline Simon.

Enfin, mes derniers remerciements vont à ma famille, ma femme et mes enfants. Ce document leur est dédié.

À Sylvaine.
À nos enfants, Elouan, Albin et Guilhem.

Table des matières

Table des figures

1

Introduction

Aujourd'hui, grâce à Internet (un réseau mondial, stable et sûr), il est possible d'interconnecter des machines du monde entier pour traiter et stocker des masses de données. Cette collection hétérogène et distribuée de ressources de stockage et de calculs a donné naissance à un nouveau concept : les grilles informatiques.

L'idée de mutualiser les ressources d'un parc informatique vient de plusieurs facteurs. Il s'agit tout d'abord d'une évolution de la recherche en parallélisme qui, après avoir étudié les machines homogènes, s'est attaquée aux environnements hétérogènes puis distribués. En outre, de nombreux efforts de développement de logiciels (par exemple Globus) ont été consentis pour utiliser les grilles. D'autre part, les besoins croissants des applications comme les simulations distribuées, la génomique (décrypthon), la bio-informatique, l'imagerie médicale, le stockage et le traitement distribué de résultats d'expériences scientifiques (par exemple en physique des particules) nécessitent l'utilisation toujours plus importante de moyens informatiques. Enfin, la nature elle-même des données a changé. Alors que pendant très longtemps le lieu de stockage et le lieu de traitement des données a été le même, la situation a complètement changé avec les infrastructures de stockage distribué (comme c'est le cas dans le projet LCG du CERN) et l'acquisition répartie des données (comme pour les applications qui gèrent des valeurs boursières de différents marchés).

La notion de grille peut avoir plusieurs sens suivant le contexte et ne fait pas l'unanimité. Dans ce document, nous appellerons grille, un système constitué de ressources *hétérogènes* et *distribuées*. Ceci regroupe aussi bien les grappes de grappes, que les environnements de type GridRPC [55], les réseaux pair-à-pair, les systèmes de calcul sur Internet, etc.

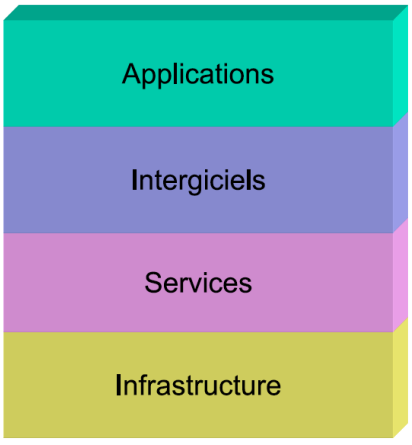


FIG. 1.1 – Modèle en couche des grilles

Pour exploiter pleinement les ressources qui constituent une grille, il est nécessaire de les virtualiser afin de rendre le plus transparent possible l'accès à celles-ci et de simplifier la tâche des personnes qui conçoivent les applications. Cette virtualisation est assurée par deux couches logiques qui s'insèrent entre la couche *applications* et la couche *infrastructure* (voir figure 1.1). La première couche est la couche *intergiciels* (en anglais *middleware*) et la deuxième couche est celle des *services*.

Le rôle de l'intergiciel est de fournir une interface (*API*) de programmation et de définir le modèle d'exécution des applications. La couche service fournit un ensemble de fonctionnalités qui

permette à l'intergiciel d'accéder aux ressources, de les contrôler ou d'obtenir des informations sur celles-ci.

Chacune de ces quatre couches logiques recèle sa propre problématique scientifique. De plus, au sein d'une même couche se pose le problème de l'interopérabilité des solutions proposées alors qu'entre les couches se pose le problème de l'intégration des solutions.

Pour la couche applications, des travaux portent sur la définition de modèles algorithmiques permettant l'écriture de programmes efficaces. En effet, les modèles algorithmiques proposés dans le cadre du parallélisme s'appliquent difficilement aux infrastructures distribuées à grande échelle. D'autres travaux visent au portage d'applications existantes dans l'objectif de passer à l'échelle [5].

La recherche sur les intergiciels porte essentiellement sur la définition du modèle d'architecture logiciel qu'il implante (GridRPC [55, 28], objets distribués [58], composants [56], pair-à-pair, etc.). Le problème du déploiement des différentes parties d'un intergiciel sur l'infrastructure est aussi un problème qui soulève de nombreuses questions.

Les intergiciels reposent sur un certain nombre de services pour accéder à l'infrastructure. Parmi ces services, citons : l'ordonnancement [13, 4], la découverte de ressources, le stockage et la persistance de données [15], l'exécution de calculs, la détermination de l'état des ressources [66], l'authentification, etc. Typiquement, un même service peut-être utilisé par plusieurs intergiciels. Pour cela il doit être suffisamment robuste et générique et son accès doit être normalisé. Il doit aussi reposer sur un ou plusieurs algorithmes efficaces.

Enfin, la couche infrastructure regroupe l'ensemble des ressources qui constitue la grille. Nous distinguons ici, trois classes de ressources : les ressources réseau, de calcul et de stockage. Les ressources réseau permettent le transfert de données. Les ressources de calcul permettent l'exécution de programmes. Les ressources de stockage permettent de sauvegarder des données. Chacune de ces ressources fait l'objet de travaux propres, et est en soi un domaine de recherche très actif (protocole réseau, base de données réparties, exécution sécurisée d'applications distantes).

L'objectif des travaux que j'ai mené depuis ma thèse est de rendre possible l'exécution *efficace* d'applications sur les infrastructures parallèles. On peut adresser le problème de l'efficacité des applications à chacun des niveaux présentés plus haut (applications, intergiciels, services, infrastructure). Cependant, nous pensons que la couche service est celle qui recèle le plus de défis scientifiques au niveau algorithmique. Ce document présente donc les problématiques scientifiques essentiellement centrées sur les modèles, les algorithmes et les protocoles pour les services de gestion des ressources. Plus précisément, nous développerons deux types de services qui doivent être particulièrement bien conçus pour atteindre une efficacité optimale. Il s'agit des services d'ordonnancement (chapitre 2), et des services de transfert de données (chapitre 3).

Une fois mis au point les modèles, les algorithmes et les protocoles, la question de leur validation se pose. Or, compte tenu de la complexité des environnements que nous étudions, il n'est pas toujours possible de prouver analytiquement les propriétés des algorithmes et des protocoles. Une validation expérimentale s'impose alors. Il en va de même pour les modèles que seule l'expérimentation peut valider. Nous détaillons dans le chapitre 4 quelle est l'importance de la validation expérimentale des modèles et des algorithmes dans les infrastructures distribuées, ainsi que les différentes méthodologies pour l'expérience dont, en particulier, Grid'5000.

2

Ordonnancement des calculs

2.1 Introduction

2.1.1 L'ordonnancement : une des clés de la performance des intergiciels

Comme nous l'avons dit dans l'introduction, un intergiciel dispose d'un ensemble de services pour exécuter une application sur une infrastructure. Parmi ces services, le service d'ordonnancement a pour rôle de déterminer l'ensemble des ressources qui vont être affectées à une application. Accessoirement, un ordonnanceur doit aussi déterminer la date à laquelle les ressources vont être utilisées par l'application.

Les performances de l'application dépendent donc beaucoup de la manière dont sont choisis la ou les ressources qui vont participer à son exécution. Ainsi le rôle de l'ordonnanceur se fera d'autant plus sentir que le taux de saturation (le rapport entre la demande en calcul et la fourniture en puissance de traitement) sera élevé.

Ceci dit, il est frappant de constater la pauvreté des algorithmes d'ordonnancement dans la plupart des intergiciels disponibles. Pendant longtemps les ordonnanceurs de DIET[1] et dans une moindre mesure de NetSolve[2] ont été très proche du "*round-robin*". Dans le Globus toolkit[3] version 4.0.2, le module pour faire de l'ordonnancement, appelé CSF (Community Scheduler Framework) est juste, comme son nom l'indique, un cadre qui définit l'API pour soumettre un job ou réserver un nœud de la grille mais pas pour ordonnancer différents jobs soumis par plusieurs clients.

Une des raisons à cette pauvreté est que la performance du service d'ordonnancement n'affecte pas le fonctionnement de l'application, alors qu'un service d'authentification ou de transfert fiable doit fonctionner parfaitement pour être utilisable. Il est donc bien compréhensible que les développements se soient d'abord portés sur ses derniers services que sur l'optimisation d'un service d'ordonnancement.

Une autre raison vient des objectifs qui ont guidés et guide encore la conception des intergiciels. Fin 2005, lors de sa présentation[4] du projet européen nextgrid[5], D. Laforenza, un des meilleurs spécialistes du domaine, définit les propriétés des grilles du futur (*transparent and reliable ; open to wide user and provider communities ; pervasive and ubiquitous ; secure and provide trust across multiple administrative domains ; easy to use and to program ; persistent ; based*

[1] Distributed Interactive Engineering Toolbox http://graal.ens-lyon.fr/~diet/
[2] http://icl.cs.utk.edu/netsolve/
[3] http://www.globus.org/toolkit/
[4] http://kathrin.dagstuhl.de/files/Materials/04/04451/04451.LaforenzaDomenico.Slides.pdf
[5] http://www.nextgrid.org

on standards for software and protocols ; person-centric ; scalable ; easy to configure and manage) sans citer la performance et l'efficacité... Si on peut comprendre qu'un intergiciel doit d'abord fonctionner avant d'être performant, il n'en reste pas moins que cette dernière caractéristique n'est pas toujours présente à l'esprit des personnes qui conçoivent ou ont conçu des intergiciels et des services.

Une dernière raison est que la conception des intergiciels s'est parfois faite dans la précipitation, au prix d'une véritable démarche scientifique basée sur la mise au point d'algorithmes et de protocoles efficaces, qui une fois implantés et validés pourraient servir de brique de base à une infrastructure globale. Par exemple, l'algorithme d'ordonnancement implanté dans NetSolve est MCT (Minimum Completion Time) [54]. Or il n'a pas été conçu pour ce cadre là (environnement distribué en mode agent-client-serveur). Par exemple, il tend à surcharger les processeurs les plus rapides au risque d'écrouler leur performance [9]. Une meilleure phase de conception aurait sans doute permis d'éviter ce problème et de mettre au point ou d'un algorithme qui alloue de manière plus efficace les requêtes sur l'infrastructure.

Naturellement, la notion de performance dépend du point de vue où l'on se place. Trop souvent les recherches en ordonnancement se placent uniquement du point de vue de l'application sans tenir compte nécessairement des autres points de vue. Or, un environnement de calcul distribué géré par un intergiciel possède plusieurs acteurs, donc plusieurs points de vue qu'il convient d'identifier et d'essayer de satisfaire.

En général, on distingue trois acteurs différents. Le *client* est l'utilisateur de la grille. Il souhaite utiliser celle-ci pour exécuter son application sur les ressources la composant. À l'autre bout, le *fournisseur de ressources* met à disposition une infrastructure pour l'exécution des programmes, le transfert ou le stockage des données. Entre les clients et les fournisseurs de ressources, on trouve un *courtier* (appelé aussi *agent* ou *broker*) qui est chargé d'organiser l'exécution des applications et la gestion des données des clients sur les ressources.

Chacun de ces acteurs a des objectifs distincts et qui sont parfois contradictoires entre eux. Plusieurs métriques permettent d'évaluer ces objectifs.

Il est intéressant de constater que beaucoup de travaux se concentrent uniquement sur les métriques utilisateurs. Cette approche est bien adaptée au contexte du calcul parallèle où les rôles sont parfois joués par les mêmes personnes et où la vitesse d'exécution de l'application (une métrique typiquement utilisateur) est le critère majeur et la raison d'être de ces systèmes. En revanche, dans notre contexte où plusieurs clients et plusieurs fournisseurs se côtoient, la conception d'algorithmes pour la gestion des ressources, pour être réellement efficace, doit concilier les différents points de vue et objectifs des différents acteurs. Ceci implique que les algorithmes mis au point dans ce contexte soient *multicritères*, pour répondre aux différents objectifs de chaque acteur. Dans la section 2.4, nous détaillerons comment mettre au point et valider des algorithmes d'ordonnancement multicritères pour les infrastructures hétérogènes et distribuées.

2.1.2 Evolution des architectures et des applications

Historiquement l'ordonnancement s'est concentré sur des architectures homogènes qui pendant longtemps ont été les seules disponibles. Or, cette dernière décennie a vu le spectre des architectures évoluer dans plusieurs directions. Une direction prise est celle de l'hétérogénéité et une autre est celle de la distribution des ressources. L'hétérogénéité des ressources implique de considérer individuellement les caractéristiques des différents composants de l'infrastructure alors que la distribution des ressources nécessite de mettre en place des paradigmes d'exécutions des applications différents du cas où les ressources sont toutes centralisées. De plus, la distribution des ressources peut impliquer un changement d'échelle en termes de dimensionnement

des infrastructures. En effet, si on s'autorise à sortir des murs d'un bâtiment pour construire un environnement de calcul en agrégeant différentes ressources, le nombre de ces ressources peut très largement dépasser celui communément utilisé lors de la conception d'algorithmes pour les environnements homogènes standards. Enfin, si le facteur d'échelle est communément vu comme positif car il accroît la puissance brute disponible, il implique parfois une dynamicité, en termes de performance et de disponibilité des ressources.

Ainsi, la nature hétérogène, distribuée, dynamique et à grande échelle des infrastructures que nous ciblons, implique de concevoir de nouvelles stratégies d'ordonnancement pour gérer efficacement l'affectation des calculs sur les ressources.

Malheureusement, la complexité d'une infrastructure regroupant toutes ces caractéristiques est telle qu'il est difficile de proposer des solutions algorithmiques complètes (prenant en compte la totalité des paramètres) et efficaces.

Pour résoudre le problème partiellement, deux approches sont envisageables :

- proposer une solution pour un cas restreint, en supposant que l'infrastructure ne possède pas toutes les caractéristiques énumérées ci-dessus. Ainsi, section 2.2.2, nous aborderons le problème de l'ordonnancement pour plates-formes hétérogènes à faible échelle sans supposer que l'infrastructure est dynamique. Dans la section 2.3.2 nous étudions le problème de l'ordonnancement pour des soumissions dynamiques dans des environnements statiques et hétérogènes ou des soumissions statiques dans des environnements homogènes soumis à des pannes et donc dynamiques,
- Laisser l'intergiciel gérer certaines caractéristiques de l'infrastructure et prendre en compte les autres caractéristiques au niveau algorithmique. Un des modèles d'intergiciel que nous avons le plus étudié est le modèle agent-client-serveur (récemment standardisé sous le nom GridRPC [55]). Dans ce cas, l'intergiciel s'occupe de gérer l'aspect dynamique et à grande échelle des ressources et l'ordonnanceur s'occupe essentiellement de l'aspect hétérogène de l'environnement (section 2.2.2).

2.1.3 Ordonnancement et méta-ordonnacement

Un de nos objectifs concernant la conception de services pour la gestion des ressources est que ceux-ci puissent être implantés dans des services aussi génériques que possible. C'est à dire qu'ils puissent être utilisés par différents intergiciels tels quels. Malheureusement, dans le cas de l'ordonnancement, la diversité des modèles d'intergiciels condamne immédiatement cet objectif.

Aujourd'hui les applications qui s'exécutent sur des infrastructures distribuées sont décomposées en blocs unitaires qui, suivant leur granularité, portent des noms différents :

- on parlera de *job* si ce bloc élémentaire est un *programme* qui peut utiliser en entrée et en sortie des données sous forme de fichiers. Ces jobs sont exécutés indépendamment et peuvent parfois être parallèles. Souvent le job et l'application sont confondus car l'utilisateur ne soumet qu'un job. Si l'application comporte plusieurs jobs, on peut regrouper ceux-ci dans un "*workflow*" qui sera interprété par un moteur et donnera lieu à une séquence de requêtes d'exécution auprès d'un courtier de ressources (*resource broker*).
- on parlera de *tâche* si le bloc élémentaire est une *fonction* ou un *service* qui est, par exemple, appelé dans un programme sous forme d'appel de procédure à distance. Dans la plupart des cas, l'application est composée d'un ensemble de tâches ayant des dépendances. Ces dépendances sont facilement modélisables par un graphe de tâches.

Que l'algorithme d'ordonnancement est à ordonnancer un workflow ou un graphe de tâches ne change rien fondamentalement. En revanche, suivant le niveau de hiérarchisation de l'infrastructure, il convient de faire la distinction entre deux grandes classes d'algorithmes d'ordon-

nancement.

Si on peut ordonnancer directement une tâche ou un job sur une unité élémentaire de calcul (typiquement un processeur dans le cas de tâches/jobs séquentiels), alors on parlera d'algorithme d'ordonnancement. En revanche on parlera d'algorithme de méta-ordonnacement pour le cas où l'infrastructure est hiérarchique et composée d'une collection de machines parallèles ou de grappes possédant chacun un ordonnanceur de type *batch*. Un méta-ordonnanceur, contrairement à un ordonnanceur classique n'alloue pas directement une requête sur une ressource mais dans la file d'attente d'un autre ordonnanceur qui choisira d'allouer cette requête sur ses ressources en fonction d'un ensemble de politiques qui lui est propre.

On le voit donc, suivant le type d'infrastructure visé, la stratégie d'ordonnancement devra prendre des décisions qui donneront lieu soit à l'exécution d'une requête sur une ressource (cas de l'ordonnancement) soit à la soumission de cette requête à un autre ordonnanceur (cas du méta-ordonnancement). Ainsi, chaque cas donnera lieu à une conception d'algorithme et à une validation différente.

2.2 Prise en compte de l'hétérogénéité

2.2.1 Nature et impact de l'hétérogénéité sur l'ordonnancement

L'hétérogénéité d'un environnement distribué se situe essentiellement à deux niveaux. Le premier niveau concerne les logiciels (services, système d'exploitation ; etc.) installés sur chacune des ressources. Le second niveau concerne le matériel où les ressources de calculs de stockage et de réseau peuvent être hétérogènes.

L'hétérogénéité en termes logiciel est essentiellement prise en compte dans le cas où on souhaiterait utiliser des services à distance. Le fait, qu'éventuellement, seul un sous-ensemble des ressources disponibles est capable d'exécuter une tâche donnée est une hypothèse couramment utilisée et elle est facilement incorporable dans des algorithmes d'ordonnancement classique[6].

Les différences en termes de capacité de stockage (mémoire, disque, etc.), sont peu abordées et modélisées lorsque l'on met au point des algorithmes d'ordonnancement pour les environnements hétérogènes et distribués. En effet, on considère souvent qu'elles sont rarement critiques en termes de performance et que l'application est conçue de manière à ne pas être confrontée aux limites de ces ressources. Dans la majorité des travaux du domaine, l'hypothèse est faite que les données utilisées ou générées peuvent être stockées sur disque et que les composants des applications occupent un espace mémoire inférieur à l'espace disponible. On peut cependant remarquer que les ressources de stockage étant finies, elles sont par nature différentes des ressources de calculs ou de réseau où les caractéristiques de ces derniers influencent les performances en termes de temps d'utilisation et pas seulement de manière binaire. Ceci rend l'algorithmique sur les ressources de stockage différente. C'est peut-être aussi une explication du peu de contribution dans ce domaine.

Les deux ressources qui rendent l'ordonnancement en milieu hétérogène si différent de l'ordonnancement en milieu homogène sont les ressources de calculs et réseau. En effet dans le cas homogène la durée d'un calcul ou d'un transfert ne dépend pas de la ressource sur laquelle elle va être effectuée, elle peut donc être calculée avant de réaliser l'ordonnancement (si par exemple on connaît le nombre d'opérations de cette tâche ou si on l'a déjà exécutée sur une des ressources). Cela permet, par exemple, de trier les tâches par durée croissante ou de calculer, à priori, le

[6]Dès sa conception, NetSolve modifie MCT pour ordonnancer des requêtes sur le sous-ensemble des ressources qui peuvent les exécuter, alors que l'algorithme ne fait pas cette hypothèse

chemin critique d'une tâche. En revanche, dans le cas hétérogène, cette durée dépend du couple tâche-processeur pour le calcul ou du couple quantité de donnée-lien pour une communication. Dans ce cas trier les tâches par ordre de durée ou calculer le chemin critique n'est plus possible avant de réaliser l'ordonnancement : les algorithmes qui utilisent cette notion ne peuvent pas directement être utilisés dans le cadre hétérogène. Ainsi les techniques d'ordonnancement en deux phases initialement proposées par Sarkar (allocation sur des processeurs virtuels puis regroupement sur les processeurs réels) [59] et qui fonctionnent très bien sur des ressources homogènes (DSC [69], DCP [49], etc.) sont difficilement transposables dans le cas hétérogène tel quel.

Les solutions classiques pour remédier à ce problème consistent soit à tester d'avantage de combinaison ressources/tâches au prix d'une complexité accrue des algorithmes ou bien à établir un ordre en fonction de la valeur moyenne ou maximale de la grandeur à considérer [41] au prix d'une imprécision d'autant plus accrue que l'environnement est hétérogène.

2.2.2 Ordonnancement des calculs sur les infrastructures hétérogènes

La prise en compte de l'hétérogénéité a été un des premiers pas vers des architectures distribuées dynamiques et à grande échelle. Les premiers travaux se sont donc d'abord concentrés sur des architectures uniquement hétérogènes avant de prendre en compte d'autres critères.

Prise en compte de l'hétérogénéité uniquement. Dans le cas homogène, le problème de l'ordonnancement de tâches parallèles consiste à affecter des tâches sur des processeurs (ou des machines), connectés par un réseau à faible échelle, en respectant deux contraintes :

- la contrainte de ressource qui interdit que plusieurs tâches puissent s'exécuter en même temps sur la même machine,
- la contrainte de dépendance qui interdit qu'une tâche puisse commencer son exécution avant que ses prédécesseurs n'aient fini leur exécution et envoyé des données éventuelles à celle-ci.

Dans le cas de l'ordonnancement de tâches sur des ressources hétérogènes ces deux contraintes doivent continuer à être respectées. La différence étant que le temps d'exécution d'une tâche ou de transfert des données dépend de la ressource utilisée. Comme nous l'avons vu précédemment, cette différence implique que l'on ne peut plus effectuer certaines opérations sur le graphe de taches (comme le calcul du chemin critique ou la durée d'exécution d'une tâche) avant le calcul de l'ordonnancement.

Plusieurs techniques ont été mises en œuvre ou adaptées du cas homogène pour remédier à ce problème :

- ordonnancement de liste : on ordonnance les tâches suivant une liste de priorité qui dépend, par exemple, de la structure du graphe de tâches en entrée de l'algorithme (HEFT [41], etc.),
- amélioration itérative : on fait un placement initial qu'on améliore jusqu'à atteindre un critère voulu.
- ordonnancement glouton : on ne revient jamais sur une décision, de placement d'une tâche (exemple : MCT [54] pour le cas online et min-min [7] pour le cas offline),
- les techniques de regroupement (*clustering* [31]), que nous avons étudié avec B. Cirou [16] (voir ci-dessous).

Aujourd'hui le problème a été bien formalisé et étudié. Sa filiation directe avec le cas homogène a permit de concevoir des algorithmes qui sont universellement reconnus pour leur efficacité (comme HEFT).

Ordonnancement pour le modèle agent-client-serveur. Si passer du cas homogène au cas hétérogène peut se faire sans utiliser un intergiciel, passer à l'échelle nécessite d'utiliser cette couche intermédiaire pour faciliter l'exécution de l'application. En effet, passer à l'échelle dans un environnement distribué et parfois dynamique implique que pour exécuter des tâches il va falloir, entre autre :

- gérer le lancement des tâches,
- prendre en compte l'enregistrement (et éventuellement la déconnexion) des ressources,
- établir un état qualitatif des ressources en termes de performance,
- mettre en œuvre des mécanisme de sécurité, d'authentification, etc...

Il s'agit d'autant de fonctionnalités qui doivent être implantées dans un intergiciel et qui peuvent être abstraites pour le service d'ordonnancement.

Il faut donc que l'ordonnanceur soit adapté au type d'intergiciel qu'il vise. Ainsi, compte tenu de la variété des intergiciels disponibles, nous restreignons ici notre propos aux intergiciels de type agent-client-serveur sur lesquels nous avons le plus travaillé.

Ce modèle est illustré par la figure 2.1 et fonctionne de la façon suivante : dans la phase d'initialisation de l'environnement, chaque serveur déclare ses services auprès de l'agent (appelé aussi *Registry* ou encore *RMS, Resource Management System*). Lorsqu'un utilisateur requiert un service, il soumet sa requête à l'agent du système. Celui-ci lui communique en retour un identifiant donnant accès à la ressource recherchée. Grâce à cet identifiant, le client contacte le serveur qui lui retournera les résultats du service exécuté.

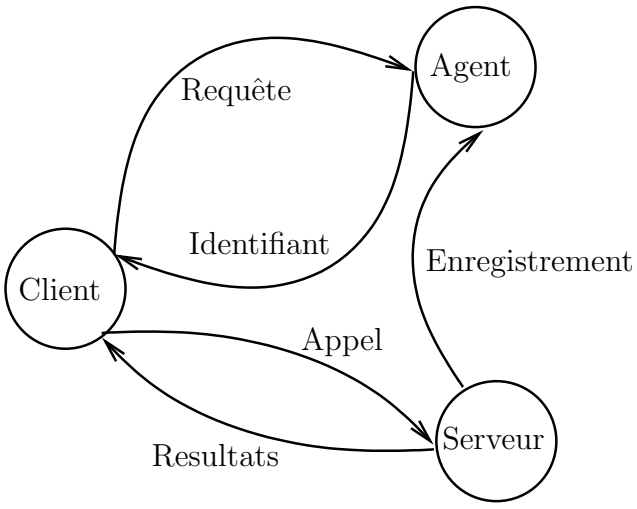


FIG. 2.1 – Modèle de base GridRPC

Le modèle GridRPC [55] est une standard proposé par le GGF[7] pour formaliser le modèle agent-client-serveur avec la différence que l'on peut créer une requête, l'allouer à une ressource et attendre un temps quelconque avant de l'exécuter, ce qui rend les décisions d'ordonnancement difficiles à prendre puisque l'environnement peut énormément changer entre l'allocation et l'exécution. Ceci limite l'intérêt du modèle GridRPC pour l'ordonnancement et c'est pourquoi nous nous limiterons au modèle agent-client-serveur original.

Dans ce modèle, les tâches sont soumises dynamiquement par plusieurs utilisateurs. Les stratégies d'ordonnancement statiques qui font l'hypothèse d'une connaissance complète de l'ap-

[7] http://www.globalgridforum.org/

plication ne sont pas applicables dans ce contexte. Il faut donc mettre en œuvre des stratégies en-ligne ("*on-line*"), qui ordonnancent les tâches/requêtes sans pouvoir anticiper sur les soumissions futures.

Une autre caractéristique importante est l'abandon de la contrainte de ressource. En effet, dans ce modèle, si au moins une ressource peut exécuter la requête, alors la requête sera ordonnancée, quel que soit le nombre de tâches en cours d'exécution sur la ressource sélectionnée. Dans l'hypothèse où plusieurs tâches s'exécutent au même moment sur la même ressource on laissera l'ordonnanceur système exécuter, en temps partagé, les processus associés à ces services. L'avantage de procéder ainsi est qu'une requête n'est jamais rejetée pour cause de surcharge de l'environnement. L'inconvénient est que si l'ordonnanceur choisit toujours la même ressource (par ce qu'elle est très rapide) celle-ci pourra s'écrouler par manque de mémoire. Plus que jamais un équilibrage de charge est nécessaire pour éviter ce problème.

Ce problème a été identifié dans NetSolve où l'ordonnanceur (MCT : *minimum completion time*), choisit parmi les serveurs possibles celui qui devrait exécuter la requête le plus rapidement. Si le nombre de requêtes est très important la charge (le nombre de requêtes allouées) de chaque serveur devient proportionnelle à sa vitesse. Pour éviter qu'un serveur ne devienne trop surchargé, ce mécanisme d'ordonnancement a été modifié par les concepteurs de NetSolve pour se rapprocher d'un simple round-robin, ce qui est clairement sous optimal.

Un autre problème important est l'évaluation de la date de terminaison d'une requête. Pour résoudre ce problème, deux hypothèses doivent être vérifiées :

1. on sait calculer la durée d'une requête sur une ressource donnée lorsqu'elle est seule à utiliser celle-ci,
2. on possède un modèle précis du partage d'une même ressource par plusieurs tâches.

L'hypothèse 1 n'est applicable qu'à une sous-catégorie de service (ceux dont la durée ne dépend que de la taille des entrées), alors que l'hypothèse 2 est beaucoup plus réaliste puisque, pour une grande majorité, les ressources partagent équitablement leur puissance entre les différentes tâches.

Ici aussi il est intéressant de noter que l'ordonnanceur initial de NetSolve n'utilisait que l'hypothèse 1 (la moins réaliste) pour effectuer ses décisions de placement, en supposant que la charge serait constante au cours de l'exécution de la requête (ce qui est faux dans le cas général). Dans ce qui suit nous montrerons comment concevoir un algorithme qui se base d'avantage sur la première hypothèse que sur la seconde.

2.2.3 Résumé de nos contributions

Avec Bertrand Cirou (stage de DEA) nous avons travaillé sur l'ordonnancement et l'exécution de graphes de tâches en milieu hétérogène. L'approche que nous avons employée est similaire aux techniques classiques de regroupement (clustering) employées en milieu homogène [16]. Comme nous l'avons détaillé précédemment, les techniques de regroupement sont difficiles à adapter au cas hétérogène car elles nécessitent de connaître la durée des tâches avant de les avoir allouées. Nous avons contourné le problème en nous basant sur le pire cas (on regroupe deux tâches sur le même processeur si on est sûr que dans le pire cas, l'ordonnancement sera plus court) et sur des critères géométriques qui permettent de maintenir les regroupement de tâches allongés (dans le sens de l'exécution).

L'avantage d'une telle approche, par rapport aux algorithmes gloutons, est qu'elle permet, au prix d'une faible complexité, d'appréhender le problème de manière globale. À notre connaissance, cette technique n'avait pas encore été utilisée en milieu hétérogène. L'heuristique mise au

point est justifiée par des résultats théoriques sur plusieurs métriques qui prennent en compte les différents aspects du problème. Nous avons proposé une validation expérimentale et une comparaison avec un des meilleurs algorithmes connus [41].

Dans le cadre de la thèse d'Yves Caniou, nous avons travaillé sur des algorithmes d'allocations d'applications décomposées en tâches pour les environnements agent-client-serveur. Nous avons étudié les limites de l'algorithme glouton MCT tel qu'il est utilisé dans NetSolve. Nous avons introduit la notion *d'historique* qui permet de mieux prédire la durée d'exécution d'une tâche sur un serveur. En effet, l'historique des soumissions couplé à un modèle de partage de ressources et à une évaluation correcte de la durée des requêtes, permet facilement de simuler l'utilisation des ressources et de prédire la fin de la requête sur chacune des ressources susceptibles de l'exécuter. Elle est beaucoup plus précise que la méthode utilisée dans NetSolve et consiste à supposer que la charge va rester constante durant l'utilisation de la ressource. Pour mettre en œuvre cela, nous nous sommes basés sur la notion de perturbation. Lorsque l'on utilise, en temps partagé des ressources, l'arrivée d'une nouvelle requête ralentie les autres. Nous appelons ce ralentissement une perturbation et notre modèle de partage des ressources permet facilement de le quantifier. Nous avons proposé des algorithmes cherchant à minimiser la perturbation d'une tâche sur un serveur tout en garantissant de bonnes performances pour la tâche elle-même. Cette approche a d'abord été testée en simulation. Parmi toutes les heuristiques étudiées, celles qui présentaient les meilleures performances ont été intégrées dans NetSolve et testées en grandeur nature [9, 10].

2.3 Prise en compte de la dynamicité

L'échelle des environnements que nous étudions, leur partage entre plusieurs utilisateurs, la nature des fournisseurs de services et de ressources impliquent une dynamicité à la fois qualitative et quantitative aussi bien au niveau de la soumission des applications que de l'infrastructure. Ceci pose des problèmes en termes de fiabilité, de robustesse, de tolérance aux fautes, etc. Prendre en compte la dynamicité de l'environnement et des soumissions est donc indispensable pour proposer des stratégies d'ordonnancement véritablement adaptées à ces contraintes.

2.3.1 D'où vient la dynamicité ?

Dans le cadre de l'ordonnancement de tâches/requêtes sur des environnements distribués à large échelle la dynamicité recouvre deux phénomènes qu'il convient de bien distinguer.

Au niveau de la soumission. Tout d'abord la soumission des requêtes et des tâches est le plus souvent dynamique, c'est à dire que la date et la nature des soumissions n'est pas connu à priori. En effet, plusieurs clients peuvent utiliser l'intergiciel pour exécuter des applications selon un modèle qui ne permet pas toujours de connaître à l'avance l'ensemble des requêtes à soumettre. De plus, même si dans certains cas on peut connaître une partie du futur et s'en servir pour optimiser l'ordonnancement, il reste qu'en toute hypothèse un intergiciel doit exécuter les requêtes des clients pour un temps, à priori, indéfini, invalidant donc toute possibilité de connaître la totalité des soumissions à venir.

En conséquence, l'algorithme d'ordonnancement doit prendre des décisions sans connaître la totalité des informations. Dans la section précédente sur l'ordonnancement pour le modèle agent-client-serveur, nous avions déjà fait cette hypothèse et les algorithmes que nous avons proposés sont des algorithmes "en ligne" ("on-line"), qui prennent en compte cette contrainte. Cependant, nous n'avons pas fait d'hypothèse sur la nature de cette dynamicité ni sur une

modélisation de celle-ci. Pourtant l'intuition nous suggère que les soumissions ne sont pas soumises au hasard absolu et que certaines règles doivent les guider. Une modélisation de ces règles doit donc permettre de concevoir des algorithmes plus efficaces et de quantifier leurs propriétés plus aisément.

Au niveau de l'infrastructure. D'autre part, les ressources qui constituent l'infrastructure sont très nombreuses et distribuées. Or le facteur d'échelle implique que les pannes sont davantage fréquentes que pour des infrastructures de moindre échelle. De plus, la distribution des ressources implique que les entités administratives qui les gèrent sont distinctes. Elles ont donc des politiques de mises à disposition et de maintenance qui peuvent varier. Ainsi, alors que dans le cas homogène centralisé il est raisonnable de supposer que toutes les ressources sont fiables et donc disponibles du début à la fin de l'exécution de l'application, cette hypothèse n'est plus réaliste dans le cadre où nous nous plaçons.

Enfin, à un niveau plus fin, la disponibilité des ressources n'est pas une variable binaire. À un niveau qualitatif, les performances des ressources peuvent varier au cours du temps car elles peuvent être utilisées par d'autres usagers ou par d'autres intergiciels. En ordonnancement, les choix dépendent de la nature et des performances des ressources à utiliser. Ceci implique donc que ces choix ne sont valides qu'à un instant donné et seulement si on est capable d'évaluer finement l'état des ressources.

2.3.2 Comment modéliser et prendre en compte la dynamicité ?

Au niveau des soumissions. La prise en compte de la dynamicité au niveau des soumissions peut se faire de plusieurs manières.

La première manière consiste à adapter le cas de l'ordonnancement en ligne classique aux infrastructures hétérogènes et distribuées. C'est ce qui est fait dans NetSolve où chaque requête est ordonnancée indépendamment des suivantes en se servant de l'état courant de l'environnement et éventuellement (comme nous l'avons proposé) de l'historique des soumissions. La théorie de l'ordonnancement en ligne de graphes de tâches permet d'avoir des bornes sur la qualité de l'ordonnancement obtenu par certains algorithmes dans le cas homogène ou hétérogène. Par exemple, il est impossible de construire un algorithme "non clairvoyant", c'est-à-dire qui ne connaît pas l'avenir, ayant un ratio d'approximation meilleur que $2 - 1/m$, pour minimiser le makespan dans le cas homogène, c'est-à-dire la même borne que les heuristiques de liste dans le cas "clairvoyant".

Cependant ces algorithmes supposent que la durée des tâches est connue. Cependant, la connaissance de cette durée n'est possible que dans certains cas. C'est, par exemple, une hypothèse réaliste pour des intergiciels où la majorité des services implantés réalisent des calculs d'algèbre linéaire plein où la durée dépend presque uniquement de la taille des entrées. Si on se place dans un cadre plus général, une telle supposition n'est plus valable. Il s'avère cependant que, dans le cas de la soumission de jobs pour l'ordonnancement par lot dans les machines parallèles, il est possible de modéliser[8] la charge de manière stochastique. Il s'agit de déterminer les lois qui régissent la soumission, la durée et éventuellement le parallélisme des jobs soumis à des environnements de traitement parallèle. Dans ce contexte, ces lois sont des lois de probabilité qui déterminent la distribution des événements à modéliser (inter-arrivée, durée, nombre de processeurs, etc.). Avec de telles lois, il n'est plus nécessaire d'avoir une information précise sur chacun des jobs pour construire un ordonnanceur dont le comportement peut être analysé

[8] lire à ce propos le version préliminaire du livre de Dror Feitelson " *Workload Modeling for Computer Systems Performance Evaluation*" : http://www.cs.huji.ac.il/~feit/wlmod/

de manière précise. Dans ce cas, il faut cependant faire la distinction entre le cas saturé et non saturé. Dans le cas saturé, la charge en entrée est plus grande que la capacité de traitement des ressources disponibles et dans le cas non saturé le système peut exécuter plus de jobs qu'il ne lui en est soumis. Pour chacun de ces cas, différentes métriques peuvent être analysées, comme le temps de traitement moyen d'un job, le nombre de processeurs utilisés, le retard moyen dû à la charge, etc.

Au niveau de l'infrastructure. Quand des ressources tombent en panne ou sont retirées brutalement de l'infrastructure, les taches en cours d'exécution sur celles-ci sont perdues. On considère généralement deux approches pour pallier à ce problème. La première vient du fait que lorsqu'une tâche est perdue, seule celle-ci doit être à nouveau exécutée. Ses prédécesseurs ont déjà fourni des résultats qui peuvent être réutilisés comme entrées de la tâche en question. Il s'agit donc de ressoumettre des tâches perdues dans le système en espérant qu'il y a au moins une ressource qui peut l'exécuter. La deuxième approche consiste à dupliquer préventivement les tâches pour que si une tâche est perdue au moins un de ses duplicats terminera son exécution correctement.

Duplication et ressoumission ont chacun des avantages et des inconvénients :

- Pour la ressoumission, le principal avantage est que l'on garanti la terminaison de la tâche tant qu'il y a au moins une ressource qui peut l'exécuter. Le principal inconvénient est que réexécuter une tâche jusqu'à ce qu'elle se termine, est coûteux en temps et peut impliquer une dégradation non bornée du temps de réponse.
- Pour la duplication le principal avantage est qu'il n'y a pas de perte dans le temps de réponse si l'infrastructure est correctement dimensionnée et peut accueillir les duplicats sans retard. Les principaux inconvénients sont que l'on consomme des ressources inutilement s'il n'y a pas de panne et qu'on ne peut pas garantir qu'une tâche sera correctement exécutée puisque tous ses duplicats peuvent échouer. La seule garantie que l'on peut fournir est d'ordre probabiliste et fonction de la probabilité de panne et le nombre de duplicats.

Il semble donc qu'utiliser la duplication ou la soumission dépende du cas où l'on se trouve. Cependant, il n'existe pas d'analyse quantitative des avantages et des inconvénients de chacune de ces solutions. Il nous semble qu'une telle analyse serait une contribution très intéressante au problème.

2.3.3 Résumé de nos contributions

Dans le cadre du méta-ordonnancement où l'infrastructure est composée d'un ensemble hétérogène de grappes homogènes, nous avons étudié le courtage aléatoire ("*random brokering*") de jobs séquentiels avec Vandy Berten et Joël Goossens de l'Université Libre de Bruxelles. Il s'agit d'un environnement où des jobs sont soumis par les utilisateurs à un méta-ordonnaceur qui réparti ceux-ci sur les files d'attente des grappes qui constituent l'infrastructure. Nous avons supposé que la soumission des jobs suit une loi de Poisson et que la durée des jobs suit une loi exponentielle. Notre stratégie consiste à allouer aléatoirement les jobs aux grappes ; la probabilité de choisir une grappe étant proportionnelle à sa puissance totale (la somme des vitesses des processeurs qui le compose). Cette approche est basée sur l'intuition que plus une grappe est puissante plus elle doit recevoir de travail [4]. Nous avons étudié les performances asymptotiques de plusieurs métriques (taille des files sur les grappes, ralentissement du temps de réponse en fonction de la charge, nombre de ressources utilisées), suivant que l'on est en régime saturé ou non. Nous avons comparé nos résultats analytiques avec des simulations pour valider nos résultats.

Dans le cadre de l'ordonnancement en temps réel sur des ressources homogènes, nous avons proposé avec Vandy Berten et Joël Goossens une approche probabiliste pour gérer la fiabilité. Dans le modèle que nous avons étudié, des tâches sont soumises à intervalles réguliers sur un système parallèle. Ces tâches sont "temps-réel", c'est à dire qu'elles doivent être exécutées avant une date limite. Une autre caractéristique est qu'elles sont susceptibles d'échouer avant leur terminaison. Formellement à chaque tâche est donc associée une échéance et une probabilité d'échec. Ce problème de fiabilité peut venir soit de pannes transitives de l'environnement (le processeur s'arrête puis redémarre) ou d'erreurs internes suite à de mauvaises données d'un capteur. Pour apporter une réponse au problème de fiabilité nous proposons d'utiliser la duplication. Chaque tâche est dupliquée un certain nombre de fois en fonction, en particulier, de sa probabilité d'échec. On peut alors calculer la probabilité que toutes les tâches (sur un intervalle fixé) puissent s'exécuter correctement. On propose alors une solution à deux problèmes symétriques [3] :

- Le problème de dimensionnement. On fixe une probabilité d'échec maximale tolérée et on cherche quel est le nombre minimal de processeurs permettant d'atteindre cet objectif.
- Le problème de fiabilité. On fixe la taille de la plate-forme et on cherche qu'elle est la plus petite probabilité d'échec possible.

2.4 Prise en compte de différents critères

Comme nous l'avons dit précédemment, dans un environnement distribué à large échelle, différents acteurs interagissent avec des objectifs différents. Dans ce contexte, pour être efficace, un algorithme d'ordonnancement doit prendre en compte ces différents critères et tenter de les optimiser. Si on regarde par exemple l'algorithme implanté dans NetSolve (MCT),DeD on constate que son but est, comme beaucoup d'algorithmes d'ordonnancement, de minimiser le temps total d'exécution d'un ensemble de tâches indépendantes. Or, originellement, il a été conçu pour ordonnancer des requêtes sur des serveurs en mode "espace-partagé" : chaque requête est exécutée l'une après l'autre alors que dans le modèle agent-client-serveur, un serveur peut exécuter plusieurs requêtes en même temps. Ceci conduit à plusieurs problèmes :

- MCT est par nature *mono-critère* et *mono-client*. En ayant comme objectif de minimiser le Makespan, MCT néglige d'autres critères comme le temps de réponse de chaque requête. De plus, dans le contexte agent-client-serveur, la notion de Makespan perd son sens puisque le nombre de requêtes n'est pas borné dans le temps. Enfin, l'intuition qui conduit à choisir le serveur qui va minimiser le temps d'exécution d'une requête pour minimiser le temps d'exécution de toutes les requêtes est fausse dans le contexte temps partagé comme l'on montré nos expériences. Ceci est d'autant plus vrai que l'agent a à ordonnancer des requêtes de plusieurs clients différents.
- MCT ne fait pas d'équilibrage de charge. Comme à chaque requête, MCT choisi le serveur le plus rapide pour effectuer la requête et il tend à choisir toujours le même (sous-ensemble de) serveur. Comme nous l'avons déjà remarqué, ceci implique que les tâches précédemment allouées sont retardées ce qui dégrade leur temps de réponse d'autant.

Pour résoudre ces problèmes, une approche qui prend en compte les différents critères de chaque acteur est nécessaire à une meilleure efficacité.

2.4.1 Les différents critères

Dans le modèle agent-client-serveur chacune des trois entités présentes a des objectifs différents :

- Un des objectif client est que son application (qu'elle soit composée d'une unique requête ou d'un ensemble de requêtes) termine le plus vite possible. Pour cela on dispose de deux métriques. Le temps de réponse de chaque requête ou le temps d'exécution du graphe de tâches (le makespan) si l'application est composée de plusieurs requêtes. Le client peut aussi avoir des contraintes de coût ou des contraintes temps réel qui peuvent aussi être mesurées par des métriques adéquates. Par exemple : rapport temps de traitement surcoût, nombre d'échéances manquées.
- Le fournisseur de service veut que ses serveurs soient le plus utilisés possible. En effet, la rentabilité d'un investissement se mesure à son utilisation. Une bonne métrique pour évaluer le débit d'un serveur est le *sumflow* qui mesure la somme des intervalles de temps pendant lesquels un serveur est utilisé. Un autre objectif est la stabilité de l'environnement qui peut se mesurer par le taux de pannes ou d'inactivité des ressources.
- Le rôle de l'agent est que chaque client soit traité de manière équitable. Il ne faut pas qu'un client, en termes d'attribution des ressources, soit sans raison plus favorisé qu'un autre. De même les petites requêtes ne doivent pas être favorisées ou défavorisées vis à vis des requêtes plus grandes. Le stretch, qui mesure le rapport entre le temps d'exécution réel de la tâche et son temps d'exécution à vide est une bonne métrique pour apprécier l'équité de traitement entre requêtes courtes et longues.
  En ce qui concerne les serveurs, l'agent doit veiller à un certain équilibrage de la charge. Deux serveurs identiques dans l'idéal doivent se voir affecter la même quantité de travail.

Il s'avère que malheureusement ces métriques ne sont pas corrélées. Optimiser le temps de réponse ne conduit pas nécessairement à un meilleur stretch. De plus, comme nous l'avons vu, l'optimisation, pour une requête donnée d'un critère, n'implique pas que ce critère sera globalement optimisé pour toutes les requêtes. Par exemple, allouer une tâche sur un serveur va ralentir celles qui sont déjà en cours d'exécution. Il faut donc prendre en compte ce phénomène si on veut optimiser le temps de réponse moyen. Il s'avère aussi que certains critères sont contradictoires. Par exemple, si on veut optimiser le temps de réponse dans un environnement très peu chargé (où l'on soumet moins de requêtes que ne peut en traiter le serveur le plus rapide), alors le serveur le plus rapide exécutera toutes les requêtes au prix d'une inéquité vis-à-vis des autres serveurs.

Si on veut comparer deux heuristiques entre elles on peut évaluer chaque métrique et les comparer une à une. On peut aussi comparer deux heuristiques en comptant le nombre de fois qu'une heuristique se comporte mieux qu'une autre. Un exemple très utilisé de cette méthode de comparaison est le *pourcentage de tâches qui terminent plus tôt*. Il s'agit de faire ordonnancer par les deux heuristiques un même ensemble de tâches et de compter le pourcentage des tâches soumises qui terminent plus tôt avec la première heuristique qu'avec la deuxième. Si ce pourcentage dépasse 50% cela signifie que, dans le cas étudié, la première heuristique est meilleure que la deuxième.

2.4.2 Ordonnancement multicritères

Dans le cas où les métriques à optimiser ne sont pas corrélées et sont même parfois contradictoires, il faut mettre en œuvre des algorithmes d'ordonnancement qui permettent de trouver un compromis acceptable entre ces différents critères. Pour y parvenir, plusieurs approches sont possibles. L'approche analytique vise à concevoir des algorithmes qui approximent plusieurs critères en même temps. L'approche expérimentale permet de vérifier qu'une heuristique satisfait plusieurs critères en même temps.

Approche analytique. Un algorithme ρ-approché est un algorithme qui fournit une solution qui est au plus ρ fois plus longue que la solution optimale quelle que soit l'entrée considérée. Cette définition implique que l'on ne considère qu'un seul critère. Il est cependant naturel d'étendre cette définition au cas où plusieurs critères sont considérés. L'idée est de construire des algorithmes qui fournissent une borne d'approximation pour les deux critères. Par exemple, on peut minimiser un critère sous la contrainte qu'un deuxième critère soit déjà minimum. Ceci n'est possible que dans le cas où la minimisation de ce dernier critère n'est pas un problème NP-hard. Ou bien, on peut construire un algorithme qui fournit directement une borne pour chacun des critères. Dans ce cas, on a un algorithme (α, β)-approché.

Approche expérimentale. La construction d'algorithmes approchés n'étant pas toujours aisée, il est parfois très difficile de trouver des bornes d'approximation de manière analytique. Une solution consiste alors à simplifier le problème jusqu'à ce que l'on trouve un algorithme approché quitte à sortir d'un cadre réaliste ou applicable. Une autre solution consiste à mettre au point des heuristiques sans aucune garantie et de les tester expérimentalement. L'expérience permet de comparer les heuristiques entre elles en évaluant les métriques relatives à chaque critère, mais ne permet pas, en général, de dire quelque chose sur le rapport d'approximation de l'heuristique.

2.4.3 Résumé de nos contributions

Dans le cadre de la thèse d'Yves Caniou nous avons travaillé sur l'approche expérimentale pour l'ordonnancement multicritères d'infrastructure agent-client-serveur [11, 12]. En nous basant sur le gestionnaire de l'historique des soumissions et sur le modèle de perturbation, nous avons été capable de proposer plusieurs heuristiques qui améliorent à la fois le temps de réponse moyen, l'utilisation des serveurs et le stretch par rapport à MCT. Notre campagne d'expériences a porté à la fois sur des graphes de tâches connectés (chaîne, stencil, etc.) et sur des tâches indépendantes. Plusieurs clients étaient connectés en même temps. Ces expériences ont été exécutées sur des environnements réels. Au total elles ont réclamé plus de 55 jours non-stop de calcul. Les deux meilleures heuristiques que nous avons étudiées sont Minimum SumFlow (MSF), qui à chaque nouvelle soumission attribue la requête au serveur qui minimise le sumflow de tous les serveurs et Minimum Lenght (ML) qui retourne l'identité du serveur telle que la somme des durées restantes des tâches en cours soit minimisée.

3

Transfert des données

3.1 Introduction

Dans le chapitre précédent, nous avons étudié comment allouer des requêtes sur des ressources de calcul. Les travaux que nous présentons ici portent sur la gestion d'un autre type de ressource : le réseau. Ici aussi nous souhaitons apporter des solutions pour rendre performant le transfert des données. La notion de performance pour le transfert de données, comme pour l'ordonnancement de calcul, recouvre de nombreux aspects. Il s'agit par exemple de notions bas niveau comme la gigue, le taux de perte de paquets, le taux de paquets hors délais, etc. ou des notions plus haut niveau comme la qualité de service, la latence ou la bande passante. En ce qui nous concerne, nous souhaitons développer des services de transfert de données pour les intergiciels. Nous considérons que le réseau, en tant qu'infrastructure (du niveau matériel à la couche transport), est donné. Nous nous situons donc à un niveau intermédiaire où la performance se mesure en termes de vitesse des transferts de bout en bout. Comme la latence et la bande passante sont les deux principaux facteurs qui influencent la vitesse de transfert, c'est sur ces deux aspects que nous allons principalement nous concentrer.

Mettre au point de tels services est très important dans le cadre où nous nous plaçons. D'une part l'échelle (en termes de distance et de taille) des environnements sur lesquels nous travaillons peut être très grande, d'autre part les données à transmettre peuvent être très volumineuses (comme nous le montrerons dans la section 3.2, les applications qui fonctionnent sur les environnements distribués gèrent des masses de données de plus en plus conséquentes). Ainsi, ces deux facteurs (échelle et volume) jouent des rôles importants pour le temps de transfert des données. Les réseaux à grande échelle ont en général un débit bout en bout plus faible et une latence plus élevée que les réseaux locaux ou les réseaux qui interconnectent une machine parallèle. De plus, la faiblesse du débit est d'autant plus visible que la quantité de données à transmettre est importante.

Dans la section 3.3 nous détaillerons deux types de services :

- Des protocoles qui définissent comment l'émetteur et le récepteur vont s'accorder pour échanger efficacement des données. C'est le cas, par exemple, de la compression adaptative.
- Des algorithmes qui organisent l'envoie des messages (l'ordre, la date d'émission, la fragmentation éventuelle, etc.) de manière à respecter des contraintes de plus bas niveau sur le débit ou le nombre de connexions autorisées à un instant donné. C'est le cas pour nos travaux sur l'ordonnancement de messages pour la redistribution.

Enfin, la façon dont sont conçus les intergiciels joue un rôle important sur la manière dont sont utilisées, stockées et transférées les données. La gestion des données par l'intergiciel a

donc un impact important sur les performances au niveau applicatif. Éviter les communications inutiles et stocker les données au plus près des lieux où elles sont utilisées, comme nous le verrons dans la section 3.4, permettent d'améliorer le fonctionnement global du système.

3.2 L'importance des données dans les applications modernes

Les infrastructures à large échelle comme les grilles de calculs sont, entre autres, une évolution des machines parallèles. Dans ce sens, elles doivent donc permettent à la fois de traiter des problèmes de plus grande taille et de résoudre des problèmes plus rapidement. Or, l'accroissement du nombre de ressources de calculs ne va pas toujours de pair avec une diminution du temps de traitement.

Une première raison est d'ordre structurel. Plus on veut utiliser des ressources qui sont disponibles à plusieurs utilisateurs, plus il faudra de temps pour arriver à les réserver en même temps puisqu'il faudra attendre que toutes les ressources soient libres en même temps pour la durée désirée. Le temps pour lancer l'exécution d'une application peut devenir très long si on souhaite utiliser toutes les ressources d'une infrastructure pour une longue période. Ce problème est encore accru lorsque l'on souhaite utiliser une grille. En effet, à cause de politiques d'administrations et de maintenance qui peuvent être différentes, des pannes, des politiques de réservation, etc., il peut être impossible d'obtenir les ressources désirées en un temps raisonnable[9]. Ce problème ouvre un certain nombre de perspectives en termes d'ingénierie, de gestion d'une communauté et de politique d'ordonnancement qui sont très intéressantes. Nous avons déjà parlé des problèmes d'ordonnancement dans le chapitre précédent, mais il s'agit aussi de problématiques qui sont au cœur du projet Grid'5000 que nous détaillerons dans le prochain chapitre.

Une deuxième raison explique qu'un accroissement des ressources n'implique pas nécessairement une diminution du temps de traitement. Cette raison a des implications directes sur les applications qu'il est raisonnable de vouloir exécuter sur une infrastructure distribuée : une application parallèle n'est pas (à taille constante des données) extensible à l'infini. En effet l'accélération (le *speed-up*) décroît dès que les ressources ne sont plus capables de traiter les données en parallèle du fait des temps de transferts et de synchronisation qui deviennent rédhibitoires. Ce phénomène est d'autant plus visible sur les infrastructures à large échelle que les performances en réseau (latence et bande passante) sont des ordres de grandeurs plus faibles que sur des machines parallèles de type cluster. Cette première remarque justifie de mener des recherches sur le transfert efficaces des données sur de telles infrastructures (ce dont nous parlerons dans la suite de ce chapitre).

Mais plus encore, cela justifie d'utiliser l'extensibilité faible (*weak scalability*), à savoir accroître la taille des données pour avoir un temps de traitement constant par processeur. Ainsi, les seules applications qui ont de l'avenir sur les infrastructures parallèles sont celles qui traitent des données réellement importantes. Plus encore, la taille de ces données doit être d'autant plus importante que le nombre des ressources est grand et que la vitesse du réseau est faible. On avait déjà pu observer cet effet pour les machines parallèles où par exemple le benchmark linpack du Top 500 doit être tourné sur des données de plus en plus grandes. Cependant, pour les environnements distribués à large échelle, les contraintes mémoires limitent rapidement la taille des problèmes que l'on peut envisager de résoudre. Si, avec le temps on peut espérer que l'augmentation conjointe de la bande passante et de la taille mémoire sur les nœuds va réduire ce problème, il n'en reste pas moins qu'un certain nombre d'applications ne sont pas destiné à

[9] Au moment où nous écrivons ces lignes, il est impossibles de réserver Grid5000 en entier à cause de pannes de durées indéterminées sur certains nœuds.

court terme à être exécutées sur une grille : les applications sur des données de petites tailles ou qui réclament beaucoup de communication du fait de leur parallélisme.

Ainsi, il s'avère que contrairement à ce que l'on avait imaginé ou voulu faire croire, les grilles ne sont pas une infrastructure universelle destinée à supplanter les machines parallèles mais bien des environnements complémentaires destinés à des applications traitant des données de très grande taille ou ayant peu de parallélisme. Or, pour les chercheurs issus du monde du calcul haute performance, ce sont les applications demandeuses en puissance CPU qui viennent naturellement à l'esprit pour savoir ce que l'on veut porter sur ces infrastructures. Il nous semble donc important d'aborder la question des applications qui peuvent traiter des grandes données et que l'on peut utilement exécuter sur une grille. Dans le contexte des applications scientifiques, on peut en distinguer plusieurs types (voir [34] Chap. 22) :

- *fouille de données*. L'astronomie utilise des données générées (par des satellites, des télescopes, etc.) et stockées de manière distribuées par différentes institutions. Cette quantité de données croit exponentiellement (elle double environ tous les ans). Le genre d'applications que traitent les astronomes visent à recouper les différentes informations d'un même objet qui sont disponibles dans les différents centres de stockage. Elles nécessitent donc des capacités de recherche en lecture sur des bases de données distribuées,
- *analyse statistique*. D'énorme quantité de données (plusieurs peta-octets par ans) seront produites par les détecteurs du LHC quand celui-ci commencera à effectuer des expériences de collisions proton-proton. Le projet LCG[10] a pour but de construire et de maintenir une infrastructure visant au stockage et à l'analyse des données ainsi produites. À terme 5000 chercheurs d'environ 500 centres devront pouvoir accéder à ces données pour les sélectionner et effectuer un traitement sur l'environnement distribué,
- *simulation et analyse*. La bio-informatique comme le docking moléculaire ou la génomique nécessite l'utilisation de données extrêmement volumineuses comparé au besoin en calcul. Cela couvre, par exemple, la conception de médicaments (où l'on teste des millions de molécules sur une seule protéine), la génomique où des données semi-structurées sont stockées pour être analysées au niveau sémantique (propriété chimique). Ici aussi les données peuvent être distribuées dans plusieurs dépôts et nécessitent une infrastructure distribuée pour leur gestion.

Notre réflexion, qu'illustrent ces exemples issues des autres sciences (mais on pourrait en prendre dans d'autres domaines), montre que les grilles sont bien adaptées à la gestion de grande masse de données. Elles sont peut-être même mieux adaptées à ces problèmes qu'à l'exécution d'applications de calcul intensif. Dans ces exemples, on voit aussi que les données sont très souvent stockées de manière distribuée. Ainsi, dans tous les cas (applications intensives en données ou en calcul), la mise au point d'un ensemble de services permettant de les transférer et de les gérer efficacement est indispensable pour obtenir une exécution performante.

3.3 Prise en compte de la latence et optimisation du débit

Il est important de comprendre que les progrès technologiques ne peuvent pas à eux seuls apporter une solution au problème de la vitesse de transfert des données. La raison principale vient d'une contrainte physique : la vitesse de la lumière est finie. Même si la vitesse de la lumière est très grande, elle donne une borne inférieure sur le temps minimum pour transmettre un paquet d'un endroit à un autre. Or, aujourd'hui en passant par Internet, la latence entre Nancy (France) et Knoxville (Tennessee, USA) est d'environ 65 ms alors que la lumière met

[10] http://lcg.web.cern.ch/LCG/

environ 20 ms pour faire les 7000 Km qui séparent ces deux villes. On voit donc bien qu'il n'y a plus grand chose à espérer en terme d'amélioration de la latence pour ce cas là.

Naturellement, les performances en terme de bande passante ne sont pas limitées de la même manière. Cependant, la vitesse d'envoi entre deux machines est malgré tout limitée par la capacité des ordinateurs à générer le flux de données (accéder au disque ou à la mémoire, empaqueter les données à travers la pile protocolaire, etc.) ou à traiter ce flux lors de la réception.

Dans tous les cas, ces limites posent des problèmes en terme de vitesse de transfert des données. Les solutions pour obtenir un temps de transfert rapide seront donc de deux ordres. Pour apporter une réponse au problème de la latence, il faut être capable de quantifier celle-ci et de construire des protocoles qui prennent en compte ce facteur. Au niveau applicatif, il faut être capable de limiter le nombre de messages en agrégeant le plus possible les messages. Pour optimiser le débit, on peut envoyer des données en parallèle évitant ainsi le goulet d'étranglement que représente éventuellement une machine seule. On peut aussi compresser les données quand cela est possible.

3.3.1 Algorithmique à gros grain et "*out of core*"

En termes de temps de communication, le coût induit par la latence est d'autant plus important que le nombre de messages est grand. En effet, à chaque communication, on paye un coût (plus ou moins fixe) avant de pouvoir transmettre le premier octet. Ceci est vrai quelle que soit la manière dont on défini la latence réseau (au niveau utilisateur, applicatif ou matériel). Ainsi, avant de s'attaquer à construire des services sophistiqués qui permettront de transférer efficacement les données, il est important d'essayer de regrouper les données à transmettre de manière à limiter le nombre de messages.

L'objectif des modèles algorithmiques à gros grain comme CGM [23], BSP [50], LogP [21] ou PRO [35] est de rendre possible la conception d'algorithmes réalistes et efficaces sur machines parallèles ou distribuées en regroupant les calculs en super-étapes et en autorisant essentiellement les communications entre ces super-étapes. Un bon exemple d'algorithme à gros grain est celui de tri de Gerbessiotis et Valiant [36].

Cette méthodologie est à rapprocher de l'algorithmique "*out-of-core*", où l'on exécute des problèmes dont la taille mémoire dépasse la mémoire physique de la machine. On utilise alors des mémoires secondaires plus lentes (comme les disques), et l'on essaye d'optimiser les accès à ces mémoires secondaires en les regroupant et en les recouvrant avec les calculs. On peut ainsi cacher en partie la lenteur d'accès aux données et obtenir des performances raisonnables pour ces cas là.

Il est donc important de voir qu'en travaillant au niveau application, il est possible de concevoir des algorithmes qui gèrent efficacement les transferts de données et donc fournissent de bonnes performances. Même si les travaux que nous présentons ici ne portent pas directement sur cet aspect nous pensons que cette approche est complémentaire avec celle qui consiste à travailler sur la couche service.

3.3.2 Redistribution de données

Transfert parallèle.

La redistribution entre ordinateur consiste, dans le cadre de l'exécution d'une application sur plusieurs machines parallèles à transférer des données d'une machine parallèle à l'autre.

Ce problème apparaît dans plusieurs cas d'utilisation dont :

– Couplage de code. Les applications de couplage de code sont composées de plusieurs programmes ou composants qui participent ensemble à la résolution distribuée d'un problème de grande taille. Chaque composant pouvant être parallèle, les phases d'échanges de données sont des redistributions. Les exemples d'applications sont la simulation distribuée multi-physique comme le projet Hydrogrid [42] ou la simulation climatique avec couplage, terre-mer-athmosphère [5].
– Exécution de tâches parallèles. Le parallélisme mixte consiste à exécuter en parallèle des tâches elles-mêmes parallèles [64]. Dans ce cas aussi, les communications entre tâches parallèles sont des redistributions.
– La persistance et la redistribution. Nous montrerons l'intérêt de mettre en place des mécanismes de persistance de données sur les serveurs pour le contexte agent-client-serveur dans la section 3.4. Si une donnée est stockée sur un serveur et qu'elle est nécessaire en entrée d'un service distant, alors la communication entre les deux serveurs est une redistribution de données dès que ces deux serveurs sont parallèles.

Le problème de la redistribution a été très étudié et se compose de plusieurs étapes [32] :
– Identification des données. Il s'agit de déterminer les données à redistribuer, leur taille et leur localisation initiale et finale.
– Génération des messages. Cela consiste pour chaque pair de nœuds à déterminer les données à échanger. En sortie de cette étape, on obtient une matrice de communication qui donne le volume à transmettre entre chaque pair de nœud.
– Ordonnancement des messages. Il s'agit de déterminer l'ordre dans lequel on va exécuter les communications, si elles vont être synchronisées, préemptées, etc.
– Exécution des communications. L'échange des données à lieu lors de cette étape en fonction des informations calculées précédemment. On peut aussi recouvrir le calcul de l'ordonnancement des messages avec son exécution, c'est-à-dire commencer à envoyer des messages sans attendre d'avoir calculé tout l'ordonnancement.

Pour effectuer une redistribution efficacement, il faut être capable de passer à l'échelle et de gérer les différentes contraintes physiques du réseau et des machines.

Ainsi la solution naïve qui consiste à rassembler toutes les données sur un même nœud avant de les transférer ne passe pas à l'échelle. Plus le nombre de nœuds est important plus le temps pour rassembler les données est grand ainsi que la mémoire requise pour stocker les messages.

On veut aussi être capable de gérer les goulets d'étranglement du réseau. En particulier, les contraintes mémoires sur les nœuds tendent à limiter le nombre de messages simultanés que l'on va s'autoriser à envoyer ou à recevoir. Le nombre total de messages que l'on peut envoyer à un instant donné peut aussi être limité. En effet, les équipements intermédiaires (commutateurs, routeurs, etc.), peuvent voir leur performance se dégrader si trop de messages circulent en même temps.

Ainsi, redistribuer des données entre machines parallèles en gérant le nombre maximal de messages qui circulent à un instant donné sur le réseau et qui sont émis ou reçus par une carte nécessite de les ordonnancer pour obtenir le temps de transfert minimal.

Prise en compte de la latence comme paramètre des algorithmes.

Pour optimiser le temps de transfert lors de la redistribution de données, on autorise souvent la préemption. La préemption consiste à permettre à un message d'être transmis en plusieurs étapes. Par exemple, le problème de l'*openshop* [38], qui est un cas particulier du problème de l'ordonnancement de messages est NP-difficile mais devient polynomial si on autorise la préemption et qu'elle n'a pas de coût. Cependant, dès que l'on prend en compte le coût de

la préemption, le problème redevient NP-difficile : la prise en compte de la latence change fondamentalement la difficulté du problème. D'une manière générale, la préemption donne plus de flexibilité pour le calcul de l'ordonnancement. Mais nous pensons cependant, que considérer que la préemption n'a pas de coût est trop optimiste. En effet, suspendre l'envoi d'un message pour le reprendre plus tard nécessite de payer au moins une fois la latence réseau. Ne pas prendre en compte la latence conduit, comme dans le cas de l'openshop, à des solutions irréalistes où les messages peuvent être découpés en petits fragments ce qui peut s'avérer très coûteux à l'exécution.

Il est cependant possible de rajouter ce paramètre latence dans les algorithmes qui traitent de la communication. Dans le cas particulier de la redistribution, prendre en compte le coût de la latence lorsqu'on autorise la préemption ne change pas le problème de classe de complexité (il reste NP-difficile). En revanche il est possible de construire des algorithmes avec garantie de performance. C'est le cas de l'algorithme proposé par Crescenzi et al. [20] pour la redistribution de données sans contrainte sur le nombre de messages qui peuvent circuler à un instant donné.

3.3.3 Compression à la volée

Le principe de la compression à la volée est d'échanger du temps de ressource en calcul par du temps de transfert. C'est-à-dire utiliser le processeur de la machine émettrice pour compresser les données et les envoyer sur le réseau et utiliser celui de la machine réceptrice pour décompresser les données.

L'idée de construire un service de compression à la volée est venu lors du projet Scilab$_{//}$ pour lequel nous avons travaillé pendant mon post-doc au LABRI. Nous avons constaté que pour exécuter des services à distance, il fallait transmettre des données (en paramètres ou en résultats) et que dans le cas du calcul matriciel où nous nous placions, ces données pouvaient être très volumineuses.

À l'époque (1999), il existait déjà des mécanismes de compression dans les outils de transfert de données comme les modems ou la commande `scp`/`ssh` (l'option `-C`). Cependant, si on regarde le manuel de `ssh` concernant l'option -C on lit : "*La compression peut être souhaitable sur les lignes modem ou les connexions lentes, mais ralentit considérablement tous les transferts si elle est activée sur les réseaux rapides*". Cette phrase explique pourquoi la compression n'est pas une option par défaut de ce genre d'outil. En fait, si nous souhaitons construire un service de compression des données à la volée qui soit universel nous pensons qu'il doit être *adaptatif*, c'est-à-dire qu'il doit autoriser ou non la compression en fonction de l'environnement de manière à ne jamais être plus lent qu'une solution qui n'utilise pas la compression. L'adaptation doit se faire en fonction de plusieurs caractéristiques :

- La vitesse courante du réseau et des machines. Pour être efficace, le temps pour compresser les données et les envoyer doit être inférieur au temps d'envoi sans compression.
- La taille des données transmise. Compresser les données va rajouter de la latence. Il y a donc une taille limite en dessous de laquelle la compression ralentie le temps de transfert total.
- La nature des données. Certaines données se compressent mieux que d'autres. Le temps de compression, à taux constant, peut aussi varier suivant les données.

Le niveau de compression doit donc être en permanence adapté en fonction de ces paramètres. La compression doit éventuellement être désactivée si les circonstances sont défavorables.

Un service de compression adaptatif à la volée, pour être le plus universel possible, doit aussi avoir les caractéristiques suivantes :

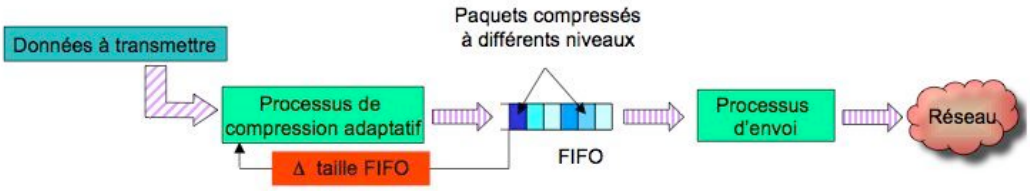


FIG. 3.1 – Principe de fonctionnement d'AdOC

- Il doit être portable. C'est-à-dire qu'il doit être construit sur des technologies standardisées. Par exemple, TCP pour la couche transport des réseaux ou POSIX pour les threads.
- Il doit être réentrant (*thread safe*). C'est-à-dire qu'un programme qui utilise les threads doit fonctionner normalement s'il utilise ce service.
- Il doit avoir une API simple et bien définie. Cela lui permettra d'être facilement implanté dans les programmes susceptibles de l'utiliser.
- Cette incorporation dans les programmes est facilitée si les fonctionnalités que le service propose ont la même sémantique que ceux qu'il améliore. Par exemple un service de compression qui remplacerait les appels système `read` et `write` devrait avoir la même sémantique, en particulier lors de la rupture de connexion ou d'un transfert partiel des données.

Finalement un service de compression doit être le plus performant possible. Il doit être capable d'améliorer les performances dans le plus grand nombre de cas possibles (même lorsque le rapport vitesse des machines sur vitesse du réseau est très bas). Le surcoût de la compression sur la latence doit être le plus limité possible lorsque l'on transmet des données de petites tailles ou que l'on utilise des réseaux rapides.

3.3.4 Résumé de nos contributions

Avec Johanne Cohen, Nicolas Padoy et surtout pendant la thèse de Frédéric Wagner, nous avons travaillé au problème de l'ordonnancement de messages pour la redistribution de données entre grappes d'ordinateurs [18, 17, 45, 46, 47, 48]. Nous avons étudié le problème de l'ordonnancement de messages en mode 1-port (le nombre de communications par nœud est limité à 1) et lorsque le nombre total de messages pouvant transiter est limité par une valeur donnée en paramètre qui correspond aux limites physiques du réseau. Nous avons formalisé ce problème par une décomposition d'un graphe biparti en couplage de taille fixée. Nous avons montré que ce problème est NP-difficile au sens fort. Nous avons proposé des heuristiques pour résoudre ce problème ainsi que des algorithmes approchés. Ces algorithmes approchés autorisent la préemption pour optimiser le temps de transfert. Cependant, de manière à limiter la fragmentation des messages lorsque celle-ci est trop coûteuse, la latence utilisateur est prise en compte en tant que paramètre. Nous avons validé expérimentalement le modèle en comparant le temps de redistribution prévu avec le temps réel. Nous avons implanté nos algorithmes et les avons testés en grandeur nature sur GRID'5000. Nous les avons comparés avec plusieurs autres approches dans le cas de la redistribution intra et inter-cluster. Enfin, nous avons généralisé le problème au cas complètement hétérogène (un nœud à plusieurs cartes réseau qui peuvent avoir des vitesses différentes) et au cas où on autorise l'utilisation du réseau local pour équilibrer les communications

Concernant la compression adaptative à la volée, nous avons écrit la bibliothèque AdOC

(Adaptive Online Compression) [43, 44]. La toute première partie de ce travail a été écrite en collaboration avec Bjorn Knutsson et Mats Bjorkmann. Avec cette bibliothèque, le niveau de compression des données transmises dépend de l'état du réseau et des ressources de calculs disponibles à chaque extrémité de la communication. L'adaptation du niveau de compression se fait en temps réel, tout au long du transfert des données. Pour des raisons d'efficacité, AdOC recouvre le transfert des données avec la compression ou la décompression (cf. Fig. 3.1). Il fonctionne sur une grande variété de réseaux et est capable d'accroître les performances pour le transfert des données jusqu'aux réseaux locaux Gbit. On peut facilement l'intégrer dans un programme existant car son API est très proche de celui des sockets et en respecte leur sémantique. Dans presque tous les cas il n'y a jamais de dégradation de la performance même pour des données incompressibles (dans ce cas AdOC désactive la compression). Le seul cas de dégradation des performances concerne les réseaux locaux Gbit Ethernet où il y a un accroissement de quelques microsecondes de la latence, visible seulement pour les données de très petite taille.

AdOC a été porté sur LINUX, Darwin, Solaris, Cygwin en 32 et 64 bits. Les gains en performance dépendent de la nature des données et peuvent aller jusqu'à une multiplication par 6 de la bande passante (pour les données ASCII). Nous avons incorporé AdOC comme service de transfert de données dans l'intergiciel NetSolve et nous avons montré qu'un gain important était possible à travers des réseaux longue distance sans qu'une perte de performance ne soit visible quand on utilise uniquement des réseaux locaux. En conclusion, il s'agit d'un service "idéal", dans le sens ou il remplace avantageusement les sockets standards et que son incorporation se fait très simplement dans un intergiciel pour le calcul distribué.

3.4 Suppression des communications inutiles

3.4.1 Persistance des calculs et redistribution

Dans les premières versions des environnements de type agent-client-serveur (comme NetSolve / GridSolve ou DIET) et plus encore dans le standard GridRPC [55], la gestion des données a été négligée. En effet dans ce paradigme, la persistance des données n'est pas gérée explicitement, contrairement aux paradigmes de programmation à base de passage de messages. Dans les environnements tels que MPI ou PVM, les données qui sont envoyées par un message (le couple `MPI_send`/`MPI_recv`), sont stockées dans l'espace d'adressage du processus qui reçoit le message jusqu'à ce que celui-ci décide de les supprimer. En revanche, dans le mode GridRPC, les données sont gérées comme dans un appel de fonction et sont détruites à la fin de l'appel du service. Ceci pose un problème flagrant d'efficacité pour ces environnements. La possibilité de garder les données localement après envoi permet comme dans les caches de les utiliser plus tard et donc autorise l'écriture d'algorithmes pouvant exploiter au mieux cette localité des données. Le paradigme de passage de messages permet aussi au processus d'envoyer des données qu'il a reçues précédemment à un autre processus, sans nécessairement passer par une entité centralisée. Dans le modèle GridRPC, les données ne peuvent être stockées et transmises qu'à partir du client : il n'est pas possible de redistribuer des données directement entre serveurs. Cela pose un problème d'efficacité puisque les données font des aller-retours entre le client et les serveurs et des problèmes de passage à l'échelle puisque le client devient un goulet d'étranglement.

Pour pallier à ce problème, différentes approches ont été proposées pour étendre le modèle agent-client-serveur :

– Dans la version 1.3 de NetSolve, le *request sequencing* a été proposé pour apporter une solution au problème des aller-retours de données superflus entre le client et le serveur. L'idée est, dans le programme client, de regrouper les requêtes en paquets (des *séquences*)

qui devront être exécutées sur un même serveur. À l'exécution les appels contenus dans une séquence sont évalués et les mouvements de données entre ces requêtes sont optimisés. Le véritable problème de cette approche est que toutes les requêtes d'une même séquence sont exécutées sur un seul serveur éliminant toute possibilité de parallélisme entre ces requêtes. De plus, il n'est pas possible d'inclure des structures de boucle dans une séquence et le graphe des appels doit être statique (il ne peut pas dépendre des résultats des requêtes).
- Les infrastructures de *stockage distribué*, permettent de stocker des données sur des serveurs répartis à différents endroits du réseau pour être accédées par des services de calcul. Ceci permet d'améliorer la performance si le stockage est proche du service mais surtout rend l'environnement plus extensible dans la mesure où le client ne limite plus la quantité de données gérées par une application. En revanche, elle ne limite pas les aller-retours entre serveur de calcul et serveur de stockage.
- L'équipe de Alexey Lastovetsky du *Heterogenous computing Laboratory* de *University College Dublin*, a proposé plusieurs approches pour résoudre ce problème. Une première approche appelée *SmartNetsolve* [6] Consiste à mettre en œuvre la persistance des données et la redistribution entre serveur en gérant le transfert des données par des proxys installés sur des serveurs. Ils transfèrent les données nécessaires à un service à partir et vers n'importe quelle location que ce soit un serveur ou le client. Cette approche nécessite de changer le code de NetSolve et c'est pourquoi ils ont proposé récemment une approche moins intrusive en créant un service de stockage qui peut être enregistré sur n'importe quel nœud. À l'aide d'un *wrapper* de l'API NetSolve les requêtes sont interceptées pour utiliser ces services de stockage et éviter les communications inutiles [51].
- Les *data handler* permettent de gérer de manière transparente l'accès aux données qu'elles soient persistantes ou pas. Dans DIET, un service de gestion des données basées sur ce mécanisme a été mis en place. Il gère, entre autre, le stockage, la sécurité, le déplacement et la réplication. Pour chaque donnée gérée par DIET, l'utilisateur crée un *handler* qui détermine le mode de persistance utilisé (stockage ou pas sur le serveur, retour ou pas au client, déplaçable d'un serveur à un autre). À chaque appel d'un service, chaque donnée est référencée par son handler. Le stockage et le mouvement de celle-ci dépendent de sa location et du type de persistance spécifié.
- Enfin, en 1999, lors de mon postdoc au LABRI, nous avons proposé de rajouter des mécanismes de persistance et de redistribution dans NetSolve. Nous détaillerons ceci dans la section 3.4.2.

Ces différents mécanismes sont de vrais progrès puisqu'ils permettent de gérer les données en les stockant sur un serveur de calcul et éventuellement en les redistribuant. Si distribuer le stockage des données est une manière efficace de passer à l'échelle pour gérer les problèmes de mémoire, cela pose un problème sur les serveurs car un client peut surcharger ses capacités de stockage. Ainsi, comme dans le cas de l'ordonnancement, il apparaît important de mettre en place des notions de qualité de service d'une part et de "contrat" d'utilisation d'autre part pour rendre ces environnements viables et utilisables à grande échelle.

3.4.2 Nos contributions

Alors que le mécanisme de *request sequencing* était le seul disponible pour gérer les problèmes des communications inutiles dans les environnements de type agent-client-serveur, nous avons proposé avec Frédéric Desprez, durant notre postdoc au LABRI un mécanisme de persistance et de redistribution des données dans de tels environnements [15, 27]. Il s'agit d'un travail de type "*proof of concept*" où nous avons modifié NetSolve pour mettre en œuvre ces mécanismes.

Au niveau du client nous avons modifié l'API pour que le programme puisse gérer de manière explicite les données et qu'il puisse exécuter des requêtes quel que soit l'endroit où elle se trouve (sur un serveur distant ou sur le client). Nous avons aussi mis en œuvre des fonctions pour rapatrier des données ou les détruire quand elles ne sont plus nécessaires.

Au niveau agent, nous avons modifié l'ordonnanceur pour qu'il prenne en compte la localisation des données et puisse optimiser l'allocation des requêtes en fonction de ces informations.

Au niveau serveur, nous avons modifié le protocole pour qu'il puisse stocker ou détruire des données, que ces données soient des paramètres de sortie ou d'entrée. Pour la redistribution, le client demande au serveur récepteur d'ouvrir une socket et transmet les informations de cette socket (le couple IP/N° de port) au serveur émetteur ainsi que l'identifiant de la donnée à envoyer. Le serveur envoie alors directement cette donnée sans qu'elle ne passe par le client.

Nous avons testé cette implantation avec un client et un agent à Bordeaux et des serveurs à Grenoble et nous avons montré que les gains en termes de performances correspondaient exactement au temps de transfert gagné grâce aux mécanismes de persistance et de redistribution.

4

Environnements pour l'expérience

4.1 L'informatique : une science expérimentale

4.1.1 L'informatique est une science ...

Un rapide tour sur Internet montre que pour la majorité de la population, l'informatique n'est pas une science, mais une technique. Si pour nous, chercheurs en informatique, une discipline, peut être à la fois une science et une technique, cette opinion (l'informatique ne serait pas une science), montre à la fois la méconnaissance du grand public de ce qu'est la recherche en informatique et plus généralement de ce qu'est une science. Cependant, il faut bien reconnaître que la question de savoir si l'informatique est une science fait débat au sein même de la communauté des informaticiens. Dans [26] Peter J. Denning montre que l'informatique remplie tous les critères d'une science. Il s'agit de rassembler et organiser un ensemble de connaissances sur notre discipline [19][11] :

> « *The discipine of computing is the systematic study of algorithmic processes that describe and transform information: their theory, analysis design, efficiency, implementation and application* »

Il note cependant, que suivant l'éducation des personnes concernées, celles-ci tendent plutôt à considérer l'informatique comme une science, une technique (" *engineering*"), ou une branche des mathématiques. Un des plus éminents avocat de l'informatique comme science est Edgser Dijkstra qui disait : " *Computer science is no more about computers than astronomy is about telescopes*"[12]. Pour lui il est clair que l'activité de chercheur est celle d'un scientifique comme les autres. Cependant d'autres scientifiques comme Dror G. Feitelson considèrent d'avantage l'informatique comme une technique [33] : " *it is an activity that leads to the creation of new tools and possibilities*"[13]. Enfin, par ailleurs, des branches de l'informatique comme la cryptologie, la complexité, les graphes, sont par certains aspects (concepts, méthodologies, rôle de la preuve, etc.) très proches des mathématiques.

Le fait qu'en anglais informatique se dise *computer science* est un des points qui pour les anglophones génère le plus de confusion puisque dans ce cas il suggère que ce domaine est celui des ordinateurs et non celui de l'information. Cependant cette confusion existe aussi en France où même pour beaucoup de personnes *informatique serait identique à la programmation.* De

[11] La discipline de l'informatique est l'étude systématique des processus algorithmiques qui décrivent et transforment l'information : leur théorie, analyse, conception, efficacité, implantation, et application.

[12] l'informatique n'est pas plus la science des ordinateurs que l'astronomie n'est celle des télescopes.

[13] C'est une activité qui conduit à la création de nouveaux outils et de nouvelles possibilités.

plus, il est clair que pour le grand public un physicien est forcément un scientifique alors qu'un informaticien est plutôt un ingénieur ou un technicien. Ainsi, il apparaît qu'il y a un manque de vulgarisation de la recherche en informatique auprès du grand public, puisque celui-ci ignore pour une grande partie l'aspect scientifique pour ne retenir que l'aspect technique.

4.1.2 . . . expérimentale

Comme le dit Peter J. Denning dans *What is experimenal computer science ?* [24] : "la science classifie la connaissance. La science expérimentale classifie la connaissance obtenue à partir d'observations." La question de l'informatique comme science expérimentale est, pour les informaticiens, identique à la question de l'informatique comme science pour le grand public. À savoir que la méconnaissance de la recherche scientifique en informatique pour le grand public est similaire à l'incompréhension de l'aspect expérimentale de cette recherche par certains collègues.

Il y a plusieurs raisons à cela. Tout d'abord, une grande partie de l'informatique (l'algorithmique, la théorie de la complexité, la logique, la théorie de graphes, etc.) est issue des mathématiques et est en conséquence une science *démonstrative* : qui consiste à construire des modèles et à prouver des théorèmes à partir d'axiomes et de définitions. Une autre raison vient de l'aspect ingénierie de l'informatique où la validation des résultats se fait par le test. En effet, dans aucun de ces cas, l'expérience planifiée, répétée, mesurée avec comme objectif l'acquisition de nouvelles connaissances n'est à la base de l'activité. Il faut cependant noté que dans le cas de l'ingénierie, l'expérience à un rôle de validation et de test qui est aussi très important.

Cependant, l'expérience est une autre voix qui permet d'acquérir de la connaissance et qui est très utile en informatique. Il s'agit de montrer la pertinence et l'intérêt de certaines hypothèses sur lesquelles les chercheurs sont amenés à travailler en les confrontant à la réalité. Le processus qui est différent de l'aspect démonstratif et de l'aspect ingénierie de l'informatique consiste à formuler des hypothèses et à les vérifier à l'aide d'expériences permettant de les valider ou de les invalider. Lorsqu'une hypothèse s'avère valide dans le cadre expérimental où elle a été testée, elle devient un modèle qui accroît notre connaissance et qui permet de prédire le comportement des phénomènes dans ce cadre. Modéliser la réalité en vérifiant des hypothèses expérimentalement est bien le propre d'une science *expérimentale* comme l'est la physique ou la chimie. Cependant, il faut répondre à deux questions avant de pouvoir conclure que l'informatique est bien une science expérimentale. La première est "*quelle est donc la réalité que l'informatique se doit de modéliser ?*" La deuxième est "*quels sont les exemples pour lesquels cette démarche expérimentale a été appliquée avec succès dans le domaine de l'informatique ?*"

Concernant la première question, comme l'informatique est la science du traitement automatique de l'information, une première réponse est donc : la réalité a modéliser est l'information. On rejoint ici la théorie de l'information, son stockage et son traitement. Cependant, une autre partie de la réalité (beaucoup plus concrète), peut être modélisée et étudiée en informatique. Il s'agit des outils qui permettent de faire ce traitement, c'est-à-dire, les ordinateurs, les réseaux, les algorithmes, les programmes, etc. Ainsi l'objet d'étude de l'informatique bien qu'artificiel n'est pas moins réel. L'informatique n'est d'ailleurs pas la seule discipline à étudier des créations humaines, c'est le cas aussi des sciences sociales. Acquérir et organiser des connaissances sur le fonctionnement et l'utilisation de ces objets est bien une science. Et, si on se souvient que George Charpak obtint en 1992 le Prix Nobel de physique pour l'invention de la *chambre à fils* nous pouvons paraphraser Edgser Dijkstra en disant : l'informatique est autant la science des ordinateurs que la physique est celle des instruments de détection.

Concernant, la deuxième question, il existe beaucoup de cas où la démarche expérimentale a été appliquée avec succès. Dans "*Performance analysis : expérimental computer science at its*

best" [25], Peter J. Denning montre que l'étude expérimentale de modèles et d'algorithmes dans le domaine de l'analyse de performance a permis des avancées considérables. Les deux exemples qu'il cite sont les travaux sur les modèles de files d'attente et sur les algorithmes de pagination.

En ce qui concerne les réseaux de files d'attente les modèles stochastiques proposés par Jackson, Gordon et Newel dans les années 50 et 60 qui ont été étendus de différentes manières et ont pu rapidement (du fait de la simplicité des formules algorithmiques qui en découle) être validés expérimentalement. Les résultats ont montrés des erreurs maximales de 5% pour l'utilisation et de 25% pour la taille moyenne de la file. Pour les algorithmes de pagination Denning rappelle que Belady montra expérimentalement à la fin des années 60 que la politique LRU est meilleure que la politique FIFO pour remplacer les pages. Ces travaux ont conduit à des résultats sur la mémoire virtuelle et la compréhension du comportement de mécanismes qui sont encore utilisés dans les systèmes modernes.

Ainsi l'expérience est une part de l'informatique et donc l'informatique est aussi une science expérimentale. Les résultats obtenus et validés par l'expérience hier sont les fondements de l'informatique d'aujourd'hui.

4.1.3 Les différents aspects de l'expérience

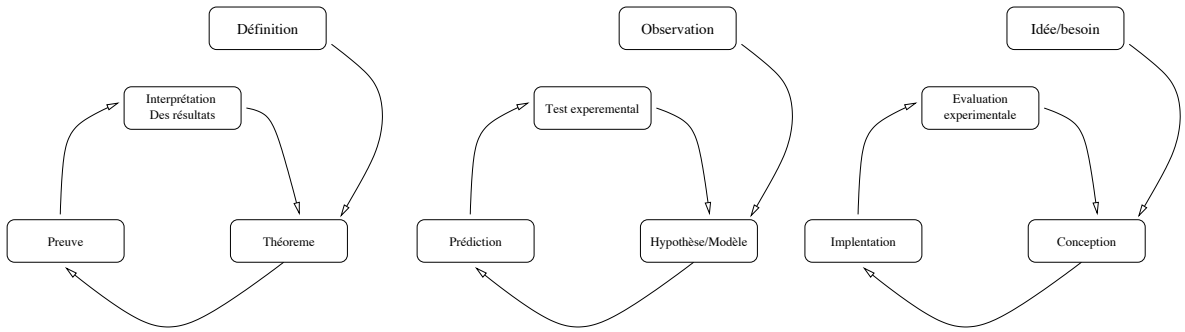


FIG. 4.1 – Les différents paradigmes de l'informatique : démonstration (gauche), modélisation (centre), conception (droite).

Précédemment, nous avons dit que suivant la sensibilité et l'éducation de chacun, l'informatique peut-être considérée comme une branche des mathématiques, une science (expérimentale) ou une technique. Ces trois aspects forment trois paradigmes qui définissent la manière dont les problèmes sont traités dans notre discipline. Ces trois paradigmes sont schématisés et illustrés dans la figure 4.1. Il s'agit des aspects démonstratifs, de modélisation et de conception qui conduisent à trois méthodologies différentes de travail. À Chaque fois, il s'agit d'un processus itératif qui permet d'améliorer l'objet que l'on cherche à construire que ce soit une théorie, un modèle ou un algorithme/programme. Dans les deux derniers cas, ce sont les expériences et les tests qui permettent cette amélioration. Il nous semble aussi important de bien préciser que ces trois méthodologies sont complémentaires. Si on prend l'exemple de la conception d'un algorithme par exemple, l'analyse de celui-ci va se baser sur une modélisation de la réalité qui devra être testée expérimentalement. À l'aide des mathématiques, on pourra ensuite démontrer des propriétés de celui-ci (complexité, ratio d'approximation, etc.). Puis, une implantation permettra de tester l'efficacité de l'algorithme sur des cas réels.

Ce qui nous intéresse ici est que notre domaine d'étude, la recherche sur les systèmes distribués à large échelle, plus encore que d'autres sous-domaines de l'informatique, contient une

approche expérimentale. En effet, les infrastructures modernes (du simple ordinateur de bureau jusqu'aux grilles), les grands programmes, les réseaux d'interconnexions, etc. sont aujourd'hui d'une complexité telle que leur compréhension analytique est quasiment impossible alors même qu'il s'agit de constructions artificielles. C'est tellement vrai aujourd'hui, que, si on donne un programme de quelques centaines de lignes, un compilateur, une description complète d'un ordinateur et de son système d'exploitation, il est impossible de dire combien de cycles processeur va prendre l'exécution de ce programme sans réaliser l'exécution elle-même ou, ce qui est pire en terme de durée, sans la simuler. Le fait que cette question pouvait encore être résolue, dans certains cas, dans les années 80 montre bien comment la complexité des environnements que nous utilisons à crue. De plus, avec la taille et l'échelle des systèmes que nous visons, des phénomènes aléatoires (pannes, utilisation partagée, etc.) rendent ces systèmes imprédictibles. Il est donc nécessaire de mesurer cet aléa et de le modéliser correctement pour prédire le comportement de l'environnement et concevoir des algorithmes efficaces.

Ainsi l'expérience est indispensable à notre domaine de recherche pour la compréhension, la modélisation et l'utilisation des environnements que nous utilisons. De plus, d'un point de vue historique, compte tenu de la complexification des objets d'étude, cet aspect de la recherche est de plus en plus présent. Comme il faut bien reconnaître qu'en termes de méthodologie, d'outil et de culture, l'informatique est très en retard par rapport aux autres disciplines expérimentale, il nous semble indispensable que les mentalités des chercheurs évoluent dans ce sens. Cette nécessité implique en particulier de définir correctement ce qu'est une expérience, quelle est son rôle et comment la réaliser. Dans la section 4.2, nous définirons le rôle et les propriétés d'une expérience en informatique. Nous donnerons des exemples d'outils et de méthodologie pour réaliser ces expériences dans la section 4.3 et enfin section 4.4, nous présenterons nos contributions dans ce domaine.

4.2 Rôles et propriétés de l'expérience en informatique

Une étude menée par Lukovicz et al. [53] au début des années 90 et reprise par Tichy un peu plus tard [63], a montré que, sur des articles publiés en informatique par l'ACM dans des journaux, entre 40% et 50% de ceux qui réclamaient une validation expérimentale n'en avaient aucune. Dans la même étude, les auteurs montraient que ce ratio tombait à 15% pour les journaux en "*optical enginnering*". Une étude statistique publiés en 1995 [70] sur la validation des résultats en informatique sur 600 articles publiés par IEEE conclus de la même manière que "*trop d'articles ne contiennent aucune validation expérimentale*", même si quantitativement cette proportion tend à décroître avec le temps (l'étude porte sur 1985, 1990 et 1995). Ces éléments montrent qu'en informatique la culture expérimentale n'est pas au niveau des autres sciences et que, bien que cela s'améliore avec le temps (du moins quantitativement), il manque une méthodologie et des outils pour réaliser des expériences.

Ainsi avant d'aborder les outils permettant de réaliser des expériences, nous allons tout d'abord montrer le rôle et l'intérêt d'une expérience et les propriétés qu'une expérience doit remplir dans notre contexte de conception d'algorithmes.

4.2.1 Rôles de l'expérience

Comme montré dans la figure 4.1, il y a deux types d'expériences. Une première catégorie permet de valider un modèle en confrontant ses prédictions avec les résultats de l'expérience. Une deuxième catégorie permet de valider et d'évaluer l'implantation d'un algorithme quantitativement. Il est important de comprendre que ces deux validations peuvent être faite en même

temps ou séparément. Par exemple, on peut valider un modèle sans avoir conçu d'algorithmes ou de programmes qui répondent à un problème dans le contexte de la modélisation. Ainsi dans [47], nous avons montré que la modélisation d'étapes de communication par un graphe biparti pour le problème de la redistribution de données donnait un temps de redistribution précis à 5% près dans le pire des cas. D'autre part, la validation et l'évaluation de l'implantation de solutions algorithmiques visent d'un côté à montrer que la solution fonctionne correctement et d'un autre côté à évaluer quantitavement ses performances et à éventuellement les comparer avec d'autres solutions. Cependant, cette phase peut aussi servir à comparer les prédictions du modèle avec la réalité et donc, dans ce cas, être aussi une expérience du premier type.

L'expérience a aussi un autre rôle tout aussi important. Dans [63], Tichy présente ainsi les bénéfices principaux de l'expérience : en testant des hypothèses, des algorithmes et des programmes, l'expérience peut permettre de construire une base de connaissances sur les théories, les méthodes et les outils utiles dans le cadre de l'étude. Les observations peuvent aussi conduire à des résultats inattendus ouvrant un nouveau champ de recherche. Enfin, les expériences peuvent conduire à des résultats négatifs et ainsi permettre d'éliminer des zones de recherche stériles, des approches erronées ou des hypothèses fausses. Elle aide ainsi à orienter la recherche dans des directions prometteuses.

4.2.2 Propriétés des expériences en informatique

Dans [1], nous avons proposé les caractéristiques que doit avoir une bonne expérience en informatique. Nous nous focalisons sur l'aspect calcul distribué mais cette approche est générale et complémentaire de l'approche démonstrative. Ainsi, dans notre domaine, une expérience doit être :

Reproductible : les conditions expérimentales doivent être conçues et décrites de manière à pouvoir être reproduites par d'autres chercheurs et doivent donner les mêmes résultats avec les mêmes entrées. Il s'agit d'un véritable défi pour la communauté informatique car cette description est souvent très élusive et parce que les environnements sont très différents d'un site à un autre.

Extensible : un rapport sur une expérience scientifique concernant la performance d'une implantation d'un système ou d'un algorithme particulier est d'un intérêt marginal s'il décrit simplement l'environnement dans lequel il a été conduit. Ainsi, la conception d'une expérience doit viser la comparaison avec des résultats passés ou futurs, des extensions avec plus de processeurs ou des processeurs différents, de plus grande quantité de données ou des architectures différentes. Plusieurs dimensions doivent être prises en compte comme le passage à l'échelle, la probabilité, la prédiction ou le réalisme.

Applicable : l'évaluation de performance est un des aspects seulement de l'expérience. La prédiction du comportement d'un programme dans le monde réel est un autre but de l'expérience. Or, l'ensemble des valeurs des paramètres et des conditions d'utilisation est potentiellement infini. Une bonne campagne d'expériences doit donc être exécuté sur un ensemble représentatif des paramètres d'entrée, des données, des usages et elle doit ainsi permettre un bon calibrage.

Révisable : quand une implantation ne se comporte pas comme prévue par le modèle, une bonne expérience doit être capable d'en identifier les raisons, qu'elles soient causées par la modélisation, la conception de l'algorithme, son implantation ou l'environnement expérimental. Il faut développer des méthodologies qui permettent de trouver et d'expliquer les erreurs de conception et indiquer des voies d'amélioration.

Dans notre cas, la validation expérimentale d'un modèle, d'un algorithme ou d'un programme sur un environnement de calcul distribué, est un véritable défi. En effet, de tels systèmes sont à très large échelle, dynamiques et partagés. Des expériences naïves sur des plates-formes réelles ne sont en général pas reproductibles alors que le caractère extensible, applicable ou révisable est très difficile à atteindre. Ces difficultés impliquent de procéder par étape en affinant le modèle et l'implantation en fonction des résultats obtenus comme décrit dans les boucles de la figure 4.1.

4.3 Exemples d'environnements

Pour conduire des expériences qui présentent les propriétés énoncées ci-dessus, il existe plusieurs manières de les réaliser. Dans chaque cas, on dispose d'outils différents avec des caractéristiques communes comme :

Le contrôle. Contrôler les conditions expérimentales est indispensable pour savoir quelles parties du modèle ou de l'implantation sont évaluées. Le contrôle permet aussi de tester et d'évaluer celles-ci de manière indépendantes. Ainsi en testant différents scenarii, le contrôle des conditions expérimentales permet d'évaluer les limites des modèles et des solutions proposées. Contrôler les conditions expérimentales nécessite donc d'être capable de configurer l'environnement ce qui, en soit, est un véritable défi.

La reproductibilité. La reproductibilité est la base du protocole expérimental. Mais c'est le rôle de l'environnement de permettre cette reproductibilité. À ce titre, un environnement autorisant une bonne reproductibilité des résultats devra donc être considéré comme meilleur qu'un environnement autorisant une plus faible reproductibilité.

Le réalisme. Les conditions expérimentales sont toujours d'une manière ou d'une autre des conditions synthétiques. C'est-à-dire qu'elles sont une abstraction de la réalité puisqu'elles ne peuvent prendre en compte tous les paramètres. Cependant, le niveau d'abstraction dépend des environnements utilisés : certains fournissent des résultats plus réalistes que d'autres.

Aujourd'hui pour réaliser des expériences, on dispose de différents types d'outils. Classiquement on distingue trois grandes classes d'environnements et d'outils pour l'expérience en informatique, avec, par ordre croissant de réalisme, la simulation, l'émulation, et l'expérience in-situ. Ces méthodologies présentent des avantages et des inconvénients différents qui font qu'elles sont plus ou moins adaptées suivant les situations.

Les simulateurs se focalisent sur certaines parties de la plate-forme et abstraient le reste du système. Ainsi la simulation permet de réaliser des expériences reproductibles et autorise de tester une large gamme de plates-formes et de conditions expérimentales. Des exemples de simulateurs pour tester et comparer des algorithmes conçus des systèmes distribués à large échelle sont : Bricks [61], GridSim [8] ou Simgrid [52]. Il est hors de notre propos de comparer les mérites relatifs de ses simulateurs et le lecteur pourra se référer à [60] pour d'avantage de détails et d'informations sur d'autres simulateurs. Il est cependant assez étonnant de constater qu'il y a très peu de résultats sur la comparaison entre des simulations et des expériences à échelles réelles. Cela pose évidemment la question du réalisme de ces environnements et montre qu'il y a encore beaucoup de travail à faire dans ce domaine. Par exemple, la validation de Bricks s'est faite en y incorporant le *Network Weather Service* (NWS) [66] et en comparant le comportement de NWS lors de l'exécution réelle d'une application et lors de sa simulation dans Bricks. La validation de Simgrid s'est faite en comparant la simulation et le résultat analytique d'un problème d'ordonnancement polynomial.

Dans certaines situations, des comportements complexes et des interactions entre les ressources distribuées ne peuvent pas être simulées. Ceci est dû à la difficulté de capturer et d'extraire tous les facteurs qui jouent un rôle durant l'exécution d'une application. Parmi ces facteurs, on peut citer : certains dispositifs des systèmes d'exploitation comme la politique d'ordonnancement des processus, la politique de gestion des pages ; certaines caractéristiques du matériel comme l'*hyperthreading*, les différentes hiérarchies de caches, les processeurs multi-cœurs ; les performances des environnements d'exécution comme les différentes versions de MPI, les différentes implantation de la norme POSIX. Les expériences in-situ visent à répondre à ces problèmes en exécutant un programme réel sur une plate-forme réelle. Typiquement des expériences sur des environnements hétérogènes sont réalisées en utilisant les stations de travail d'un laboratoire. Cependant, ces machines sont souvent partagées par d'autres utilisateurs ce qui rend la reproductibilité difficile à atteindre. Pour apporter une réponse au problème de la reproductibilité, il est possible de construire des environnements dédiés à l'expérimentation in-situ. Parmi ces environnements, on trouve : Das-3 [22], Grid-explorer [40], Grid'5000 [39, 37], ou Planet-Lab [57]. Nous reparlerons de Grid'5000 dans la suite, mais avant cela il faut remarquer que le contrôle et la configuration de ses environnements est très délicats. En général ses environnements ne sont pas configurables. C'est la cas de Plantet-Lab mais un peu moins de Grid'5000 où on peut changer toute la pile logicielle. Ainsi, l'hétérogénéité de ces plates-formes est fixe. De plus, comme les machines qui constituent ces environnements sont souvent achetées en même temps, le niveau d'hétérogénéité est donc faible. Or, contrôler l'hétérogénéité est indispensable car les environnements distribués pour lesquels nos solutions sont conçues sont, par définition, hétérogènes. Ainsi l'impossibilité de réellement configurer ces environnements pour l'expérimentation in-situ rend les expériences difficiles à appliquer dans d'autres contextes.

D'un point de vue du réalisme, entre la simulation et les expériences in-situ, on trouve l'émulation. L'émulation vise à construire un environnement synthétique (comme dans le cas de la simulation) pour y exécuter de vraies applications (comme dans le cas des expériences in-situ). Parmi les émulateurs d'environnements distribués on trouve MicroGRID [68] qui permet d'exécuter sur un ou plusieurs processeurs un ensemble de machines virtuelles qui exécutent l'application testée. La base des émulateurs est donc les machines virtuelles (comme Xen [67] ou QEMU [30]) qui ont fait d'énormes progrès (en termes de performance et de configuration) ces dernières années. Les machines virtuelles permettent d'exécuter, de manière isolée, différents environnements. Il est aussi possible de les faire communiquer entre elles en les connectant logiquement par un réseau. Cependant, toutes seules, les machines virtuelles ne permettent pas de contrôler l'hétérogénéité de l'environnement. Configurer cette hétérogénéité nécessite de mettre en place des mécanismes qui permettent de changer les caractéristiques matérielles de l'environnent comme la vitesse du processeur, la quantité de mémoire disponible, la vitesse et la latence du réseau. Evidemment, ces caractéristiques étant fixées par le matériel, si on veut les changer, il y a deux solutions. La première consiste à mettre à jour une partie du matériel (mais c'est coûteux et permet seulement un contrôle très faible de l'environnement). La seconde solution consiste à dégrader, de manière logicielle, le matériel en ralentissant par exemple la vitesse du processeur. L'avantage est de permettre le test d'une large gamme de configuration possible, mais cela pose le problème du réalisme. C'est cette dernière approche que nous avons proposé dans Wrekavoc [14] et qui sera détaillée dans la prochaine section.

Avant de présenter nos contributions dans ce domaine, nous résumons et comparons les différentes approches présentées dans la table 4.1. Les différents points étudiés sont :

- la capacité d'exécuter une application réelle ou juste de simuler son exécution.
- Le niveau d'abstraction de l'environnement. Moins il y a d'abstraction plus le réalisme est grand et meilleur est la confiance que l'on peut avoir dans les résultats obtenus.

Méthodologie	Simulation (Simgrid)	Émulation (Microgrid)	In-Situ (Env. homogène.)	Dégradation (Wrekavoc)
Application réelle	Non	Oui	Oui	Oui
Abstraction	Très haute	Haute	Non	Basse
Temps d'exécution	Accélération	Ralentissement	Identique	Identique
Repliement des proc.	Obligatoire	Possible	Non	Non
Hétérogénéité	Contrôlable	Contrôlable	Non	Contrôlable

TAB. 4.1 – Comparaison des différentes méthodologies expérimentales

– La vitesse d'exécution. L'émulation tend à ralentir l'exécution alors que la simulation tend à l'accélérer.
– La possibilité de replier plusieurs processeurs sur un seul. Ceci permet d'exécuter une application parallèle sur moins de processeurs (éventuellement un seul) que dans la réalité.
– La gestion de l'hétérogénéité. Est-il possible d'avoir de l'hétérogénéité ? Est-elle contrôlable ?

En conclusion, nous voyons qu'aucune de ses approches n' a que des avantages ou des inconvénients. Elles sont complémentaires, et cela signifie que selon les objectifs expérimentaux recherchés il est nécessaire de déterminer quelle est celle qui est la plus adaptée.

4.4 Nos contributions

Nous avons travaillé à la mise en place du site de Grid'5000 [37] sur Nancy. L'objectif de Grid'5000 est de construire un instrument pour réaliser des expériences en informatique dans le domaine du calcul distribué à grande échelle. Cette plate-forme regroupe 9 sites répartis sur le territoire national. Chaque site héberge une ou plusieurs grappes de processeurs. Ces grappes sont alors interconnectées via une infrastructure dédiée fournie par Renater. À ce jour, Grid'5000 est composés de 9 sites (Lille, Rennes, Orsay, Nancy, Bordeaux, Lyon, Grenoble, Toulouse et Nice – voir Fig. 4.2). Début 2007, Grid'5000 regroupe plus de 2500 processeurs et près de 3500 cœurs.

Les caractéristiques principales de Grid'5000 sont :

– Un haut niveau de confinement. Il est impossible d'accéder à un nœud de Grid'5000 depuis Internet et vis versa.
– Une transparence d'accès à l'intérieur de Grid'5000. Une fois connecté à l'infrastructure tous les nœuds de Grid'5000 sont accessibles directement.
– Des mécanismes de réservation permettant de réserver et d'organiser des expériences. OAR, l'outil de réservation de Grid'5000 est parfaitement intégré à l'environnement et permet aux utilisateurs de sélectionner les nœuds dont ils ont besoin pour leur expérience.
– Une boîte à outil permettant de configurer les nœuds. Kadeploy permet de déployer une image sur chacun des nœuds correspondant à l'expérience que l'utilisateur souhaite réaliser et lui donnant toute latitude pour la configurer.

À Nancy, avec Martin Quinson et Xavier Delaruelle, nous avons travaillé à l'acquisition, à l'installation et à la mise à disposition des deux grappes du site. Les deux machines de Nancy sont : une grappe de 47 AMD Opteron 64 bits biprocesseurs et une grappe de 120 Intel Xeon 64 bits biprocesseurs bicœurs. Dans les deux cas, les réseaux d'interconnexion sont du giga-ethernet. D'un point de vue scientifique, de nombreuses expériences sont réalisées par des utilisateurs du Grand-Est de la France. Ces utilisateurs sont rattachés au site de Nancy et viennent des

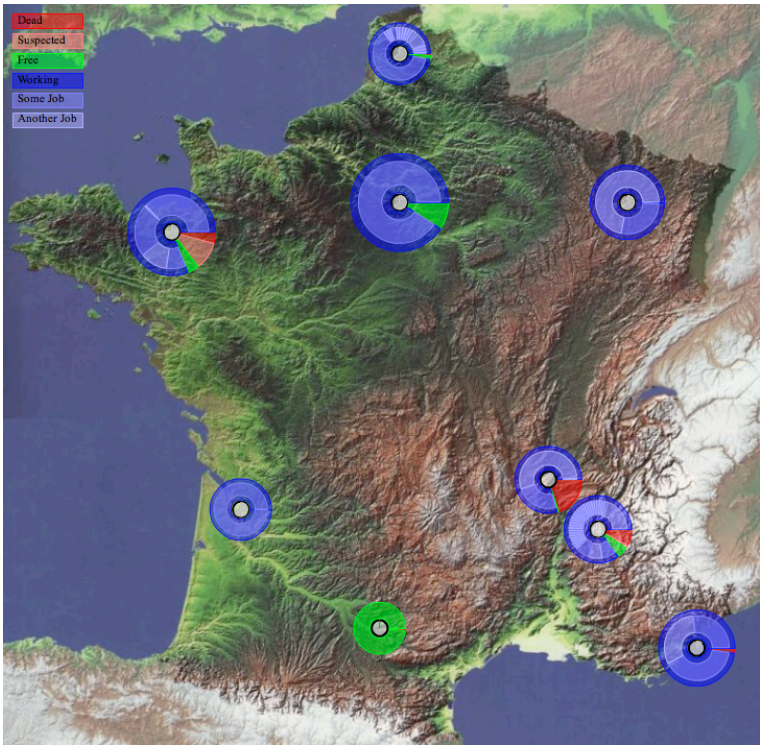


FIG. 4.2 – L'état du site Grid'5000 le 27 mars 2007 à 10h25

grands laboratoires des villes du Grand-Est de la France : Nancy (LORIA, IECN, eDAM), Metz (Supelec et LITA), Reims (LICA-CReSTIC), Strasbourg (ICPS-LSIIT, Observatoire de Strasbourg), Besançon (LIFC). Les différentes recherches menées sur ce site recouvrent des travaux allant des réseaux (analyse de plates-formes de management), aux applications (docking moléculaire) en passant par l'algorithmique (ordonnancement, transfert de données, calculs out-of-core), les intergiciels ou le calcul numérique.

Avec Louis-Claude Canon, Olivier Dubuisson, Jens Gustedt et Marc Thierry, nous avons travaillé à un outil de contrôle et de configuration de l'hétérogénéité de grappes de PC appelé Wrekavoc [14] – http://wrekavoc.gforge.inria.fr. L'objectif est de transformer une grappe ou un ensemble de grappes homogènes en grappes hétérogènes et de contrôler cette hétérogénéité. Pour cela nous dégradons les caractéristiques suivantes : vitesse CPU, quantité de mémoire disponible, bande passante et latence réseau. Cette dégradation se fait de manière logicielle sans changer le matériel ni redémarrer les nœuds des grappes en question. Wrekavoc permet de configurer plusieurs îlots indépendants et de les relier entre eux par un réseau logique. La dégradation du réseau est effectuée grâce à Traffic Control (TC) [62]. En ce qui concerne la dégradation de la vitesse du processeur on utilise soit CPU-freq (un module du noyau linux), soit un processus qui prend du temps processeur, soit un système d'ordonnancement utilisateur pour suspendre et activer les processus en fonction de la dégradation demandée. Enfin, pour la mémoire, la dégradation est réalisée en allouant une partie de la mémoire et en punaisant celle-ci en mémoire physique. Wrekavoc est disponible sous le gforge de l'INRIA à l'adresse http://wrekavoc.gforge.inria.fr.

5

Synthèse et perspectives

5.1 Résumé des contributions

Depuis la fin de la thèse nous avons travaillé sur des environnements distribués et hétérogènes. Ces environnements prennent une place de plus en plus importante pour exécuter des applications scientifiques. En effet, les applications réclament toujours d'avantage de puissance de calcul ou de capacité de stockage. Ces infrastructures sont aussi très adaptées aux applications traitant des données volumineuses comme la bioinformatique, la physique des particules ou l'astronomie.

Dans ce cadre, nos recherches s'inscrivent dans le développement, l'analyse et l'expérimentation de modèles, d'algorithmes et de protocoles permettant la réalisation de services de gestion des ressources pour de tels environnements. Plus précisément, les différents thèmes de recherche sur lesquels nous avons travaillé sont :

- l'ordonnancement des calculs. Les problèmes que nous avons abordés concernent :
  1. la prise en compte de l'hétérogénéité avec la mise au point d'heuristiques pour ordonnancer des graphes de tâches ou des applications en mode GridRPC,
  2. la prise en compte de la dynamicité avec la mise au point de modèles stochastiques pour l'ordonnancement et de stratégies probabilistes pour gérer la fiabilité,
  3. La prise en compte de plusieurs critères avec la réalisation et l'étude de différentes heuristiques permettant de trouver un bon compromis entre les besoins des différents acteurs d'un environnement agent-client-serveur.
- le transfert de données. Pour ce point nos contributions sont doubles :
  1. nous avons travaillé à l'optimisation du temps de transfert pour le problème de la redistribution de données et à la compression adaptative à la volée,
  2. en développant des mécanismes de persistance des données et de redistribution, nous avons travaillé à la suppression des communications inutiles dans le contexte GridRPC.
- le dernier aspect de nos recherches est orthogonal aux précédents et concerne la validation expérimentale des solutions proposées. Il s'agit d'un travail plus récent, mais qui nous semble néanmoins très prometteur. Dans ce domaine, notre contribution est double : un environnement d'émulation de l'hétérogénéité et notre participation à Grid'5000 en particulier le site de Nancy.

5.2 Perspectives de recherche

Les travaux développés jusqu'ici ouvrent de nombreuses perspectives que nous détaillons dans le projet suivant. Il s'agit d'un ensemble de perspectives en continuité avec les recherches précédentes et qui sont des sujets de thèses potentiels. L'objectif de ce projet est de rendre possible l'exécution *efficace* d'applications sur les grilles. Dans la suite de ce que nous avons présenté jusqu'ici, il s'agit de travailler au développement de modèles et d'algorithmes pour les services de gestion des ressources en s'attaquant aux problèmes nouveaux qui nous sont apparus comme très important aux cours de ces dernières années.

Une fois mis au point les modèles et les algorithmes, la question de leur validation se pose. Nous continuerons donc à promouvoir la validation expérimentale de nos résultats que ce soit par simulation, émulation ou sur plates-formes réelles.

5.2.1 Algorithmes pour la gestion des ressources

Les algorithmes pour la gestion des ressources que nous proposons dans ce projet se situent dans la couche service comme décrite dans l'introduction. Nous souhaitons concevoir des solutions aussi indépendantes que possibles des intergiciels des systèmes et des technologies. En particulier, cela signifie que les pistes évoquées ici pourront être intégrées dans différents intergiciels et implantées de différentes manières.

Ordonnancement multicritère

Dans une grille, plusieurs acteurs interviennent, chacun ayant ses propres objectifs. Classiquement on distingue trois acteurs différents. Le *client* est l'utilisateur de la grille. Il souhaite utiliser celle-ci pour exécuter son application sur les ressources la composant. À l'autre bout, le *fournisseur de ressources* met à disposition celles-ci pour l'exécution des programmes, le transfert ou le stockage des données. Entre les clients et les fournisseurs de ressources, on trouve un *courtier* (appelé aussi *agent* ou *broker*) qui est chargé d'organiser l'exécution des applications et la gestion des données des clients sur les ressources. Chacun de ces acteurs a des objectifs distincts et qui sont parfois contradictoires entre eux.

À long terme nous souhaitons étudier comment offrir aux différents acteurs d'une grille une garantie sur les critères qui les concernent. Pour cela, nous proposons, dans un premier temps, d'étendre les premiers résultats que nous avons obtenus pour l'ordonnancement dans le contexte GridRPC.

Deux directions de travail sont possibles. Tout d'abord, nous souhaitons lever l'hypothèse selon laquelle la plateforme est utilisée en exclusivité par l'environnement. Dans un premier temps, il faudra modifier le HTM pour qu'il puisse prendre en compte des informations dynamiques de la plateforme (comme la présence d'autres tâches sur les ressources). Dans un deuxième temps, il sera nécessaire de changer les stratégies d'ordonnancement pour prendre en compte ses nouvelles informations. La deuxième direction que nous souhaitons prendre consiste à mettre en œuvre des algorithmes avec une garantie sur les différents critères évoqués ci-dessus. Pour cela, nous pourrons nous inspirer des travaux de l'équipe de D. Trystram [29] et de E. Bampis [2] sur le sujet tout en nous basant sur les résultats expérimentaux obtenus lors de la thèse de Y. Caniou [13]. Nous souhaitons ensuite étendre ces résultats à d'autres types de grilles que le GridRPC. En particulier nous viserons les grilles basées sur Globus puis des systèmes totalement distribués comme les réseaux pair-à-pair. Ceci nécessitera un changement des modèles, mais nous pensons que les techniques employées resteront valides.

Gestion de la dynamicité

Souvent, une des caractéristiques des grilles est la dynamicité de l'environnement. Cette dynamicité s'exprime en termes de volatilité des ressources (une ressource peut apparaître ou disparaître aléatoirement) et en termes d'imprédictibilité de la charge (la soumission et la durée des tâches n'est pas forcément connue à l'avance).

Notre objectif à long terme est de permettre d'exécuter efficacement et de manière transparente des applications dont on ne connaît pas la durée de manière certaine, en offrant une fiabilité garantie même en présence de fautes.

Dans un premier temps, il faudra modéliser les soumissions et la volatilité des ressources. Il s'agit de trouver et de paramétrer les lois de probabilité qui modélisent ces phénomènes. Pour cela, nous nous baserons sur les traces des ordonnanceurs et méta-ordonnanceurs qui exécutent des jobs sur des plates-formes de production comme la grille EGEE et sur des plates-formes de grilles expérimentales comme Grid'5000.

Ensuite, il faudra travailler à la mise au point de stratégies d'allocations stochastiques des tâches permettant de prendre en compte efficacement la dynamicité de l'environnement. Le cadre applicatif visé sera Grid'5000, qui est une plate-forme hiérarchique constituée d'un assemblage de plusieurs grappes. On utilisera les modèles définis plus haut pour les appliquer au méta-ordonnancement : l'algorithme alloue les tâches à des grappes qui sont ensuite ordonnancées localement entre elles.

Une troisième étape consistera à étendre ces travaux pour mettre en place de stratégies tolérantes aux pannes. Si l'option de duplication (qui s'apparente à de la prévention) et l'option de sauvegarde-redémarrage (qui s'apparente à de la guérison) ont chacune des avantages et des inconvénients, elles sont mal évaluées de manière qualitative. Il faudra donc procéder à cette évaluation en se basant sur les modèles et les algorithmes élaborés dans les deux étapes précédentes. Les métriques seront le temps d'exécution de l'application, sa probabilité de terminaison correcte, l'utilisation des ressources, le temps de réponse, etc.

En se basant sur ces résultats, les travaux pourront se poursuivre par la mise au point de stratégies fiables pour l'exécution distribuée d'applications sur des plates-formes à large échelle dynamiques et par l'évaluation de ces stratégies sur l'environnement Grid'5000. Au final, on disposera d'un ordonnanceur qui sera à la fois fiable (il tolérera les pannes) et capable de prendre en compte la dynamicité de l'environnement.

Redistribution et communication collectives pour les grilles

Dans la thèse de F. Wagner [65], nous avons proposé plusieurs solutions au problème de la redistribution entre grappes d'ordinateurs. Une redistribution est alors modélisée par un graphe biparti et une solution au problème est une décomposition de ce graphe respectant certaines contraintes (par exemple, dans le cas le plus simple, chaque sous-graphe de la décomposition doit être un couplage).

Jusqu'à présent nous n'avons étudié que le cas où la redistribution se fait entre deux grappes, où le débit du réseau ne varie pas dynamiquement et où la redistribution est connue entièrement au début de l'algorithme.

Nous proposons donc, d'étudier le cas où plusieurs grappes prennent part à la redistribution. Cela nécessite de changer les modèles (les graphes qui modélisent les communications ne sont plus bipartis) et de mettre au point des algorithmes qui s'exécutent de manière distribuée. Dans ce contexte, la problématique du passage à l'échelle devient primordiale. Nous voulons aussi traiter le cas dynamique (en termes de débit réseau et de la connaissance du schéma de redistribution).

Dans un premier temps, on pourra réévaluer les facteurs d'approximation des algorithmes déjà existants puis, dans un deuxième temps, appliquer les techniques d'ordonnancement dynamique à ce contexte.

Un autre problème vient du fait que ces techniques sont centralisées et que le temps de calcul de l'ordonnancement peut être trop long par rapport au temps effectif de transfert. Une étude intéressante consisterait à mettre au point des algorithmes de communication qui soient à la fois décentralisés et rapides. Ainsi, il ne serait plus nécessaire de regrouper les informations sur un seul nœud qui effectuerait le calcul de l'ordonnancement : on gagnerait en extensibilité de l'algorithme. Les applications candidates sont par exemple les communications collectives de MPI comme le *all-to-all*, le *gather*, le *scatter*, etc.

5.2.2 Validation expérimentale

Comme nous l'avons montré dans ce document, l'informatique, dans le contexte des grilles de calcul, est une science expérimentale. De plus, une *bonne expérience* doit répondre à plusieurs critères. Elle doit être reproductible, elle doit permettre d'extrapoler des résultats avec d'autres paramètres et à différentes échelles. Enfin, si les résultats ne sont pas conformes aux prédictions, on doit pouvoir en identifier les raisons (mauvaise modélisation, mauvaise conception algorithmique, mauvaise implantation, etc.).

Nous souhaitons donc continuer à travailler à la validation expérimentale des solutions que nous proposons. Il s'agit d'un défi particulièrement important concernant les grilles.

Pour cela nous comptons renforcer notre participation au développement de Grid'5000 en fournissant des outils et des environnements qui permettent de réaliser des expériences dans les meilleures conditions. En particulier, nous souhaitons relever le défi concernant l'émulation de l'hétérogénéité.

À ce jour, le prototype dont nous disposons (wrekavoc) est fonctionnel mais un certain nombre de problèmes reste à régler. En particulier, la dégradation de la vitesse du CPU dans le cas où les applications sont multi-threadées et où les processeurs ont plusieurs cœurs est mal gérée. Pour le réseau, Wrekavoc peut finement dégrader les communications point-à-point. Toutefois, lorsqu'une carte fait plusieurs communications simultanées (comme dans le cas des communications collectives dans MPI), la dégradation est moins précise.

Un autre objectif est de travailler à la mise en place, d'un environnement dynamique où la puissance CPU, la quantité de mémoire disponible et l'occupation du réseau varient au cours du temps suivant un schéma prédéfini et contrôlé par l'expérimentateur. Pour cela, une première étape consistera à nous inspirer de l'injection de traces telle qu'elle est modélisée et réalisée dans le projet SimGRID (http://simgrid.gforge.inria.fr/) qui est en partie réalisée dans notre équipe. Une deuxième étape consistera à injecter ces traces dans wrekavoc en assurant qu'elles sont synchronisées et qu'elles impactent de manière réaliste les applications.

Les environnements distribués à large échelle peuvent êtres constitués de plusieurs milliers (voire des centaines de milliers) d'unités de calcul et de stockage. Ces unités peuvent tomber en panne ou être retirées de l'environnement par leur propriétaire. En cours d'exécution, d'autres unités peuvent apparaître et être mises à disposition des applications. Cette dynamique doit donc être prise en compte lors de la conception de solutions permettant d'utiliser ces infrastructures à large échelle. Toutefois, l'objectif d'une plate-forme d'expérimentation comme Grid'5000 est d'être le plus stable possible, et n'offre pas par exemple une volatilité comparable à celle des systèmes pair-à-pair. Il est donc vital d'être capable de contrôler et de reproduire les fautes qui peuvent survenir dans les environnements d'exécution visés afin de tester et d'expérimenter ces solutions dans des conditions les plus réalistes possibles. Nous souhaitons donc étendre Wrekavoc

pour qu'il prenne en compte la dynamicité de l'environnement en étant capable d'injecter des fautes suivant des modèles prédéfinis.

Bibliographie

[1] Equipe AlGorille. Algorithms for the Grid. INRIA Research proposal, July 2006.

[2] Eric Angel, Evripidis Bampis, and Laurent Gourvis. Approximation results for a bicriteria job scheduling problem on a single machine without preemption. *IPL*, 94(1) :19–27, 2005.

[3] V. Berten, J. Goossens, and E. Jeannot. A Probabilistic Approach for Fault Tolerant Multiprocessor Real-time Scheduling. In *14th Workshop on Parallel and Distributed Real-Time Systems*, Island of Rhodes, Greece, April 2006.

[4] V. Berten, J. Goossens, and E. Jeannot. On the Distribution of Sequential Jobs in Random Brokering For Heterogeneous Computational Grids. *IEEE Transactions on Parallel and Distributed Systems*, 17(2) :113–124, 2006.

[5] F. Bertrand, R. Bramley, D. Bernholdt, J. A. Kohl, A. Sussman, J. W. Larson, and K. Damevski. Data Redistribution and Remote Method Invocation in Parallel Component Architectures . In *IEEE IPDPS*, 2005.

[6] Thomas Brady, Eugene Konstantinov, and Alexey Lastovetsky. SmartNetsolve : High-Level Programming System for High Performance Grid Computing. In *3rd High Performance Grid Computing Workshop*, Island of Rhodes, Greece, May 2006. In conjuction with IPDPS 2006.

[7] T. D. Braun, S. Ali, H. J. Siegel, and A. A. Maciejewski. Using the min-min heuristic to map tasks onto heterogeneous high-performance computing systems. In *Proceedings of the 2nd Annual Los Alamos Computer Science Institute (LACSI)Symposium*, Sante Fe, NM, USA, Oct. 15–18 2001.

[8] Rajkumar Buyya and M. Manzur Murshed. GridSim : A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *CoRR*, cs.DC/0203019, 2002.

[9] Y. Caniou and E. Jeannot. New Scheduling Heuristics in the Client-Agent-Server Model. In *IEEE Heterogeneous Computing Workshop (HCW'03)*, Nice, France, April 2003.

[10] Y. Caniou and E. Jeannot. Efficient Scheduling Heuristics for GridRPC Systems. In *QOS and Dynamic System workshop of IEEE ICPADS (International Conference on Parallel and Distributed Systems) conference*, pages 621 – 630, New-Port Beach, California, July 2004.

[11] Y. Caniou and E. Jeannot. Experimental Study of Multi-Criteria Scheduling Heuristics for GridRPC Systems. In *ACM-IFIP Euro-Par*, Pisa, Italy, September 2004.

[12] Y. Caniou and E. Jeannot. Multi-Criteria Scheduling Heuristics for GridRPC Systems. *International Journal of High Performance Computing Applications*, 20(1) :61–76, spring 2006.

[13] Yves Caniou. *Ordonnancement sur une plate-forme de métacomputing.* Thèse d'université, Université Henri Poincaré, December 2004.

[14] L.-C. Canon and E. Jeannot. Wrekavoc a Tool for Emulating Heterogeneity. In *15th IEEE Heterogeneous Computing Workshop (HCW'06)*, Island of Rhodes, Greece, April 2006.

[15] E. Caron, B. Del-Fabbro, F. Desprez, E. Jeannot, and J.-M. Nicod. Managing data persistence in network enabled servers. *Scientific Programming Journal*, 13(4) :333–354, 2005.

[16] B. Cirou and E. Jeannot. Triplet : a Clustering Scheduling Algorithm for Heterogeneous Systems. In *IEEE ICPP International Workshop on Metacomputing Systems and Applications (MSA'2001)*, Valencia, Spain, September 2001.

[17] J. Cohen, E. Jeannot, N. Padoy, and F. Wagner. Message Scheduling for Parallel Data Redistribution between Clusters. *IEEE Transactions on Parallel and Distributed Systems*, 17(10) :1163–1175, October 2006.

[18] Johanne Cohen, Emmanuel Jeannot, and Nicolas Padoy. Messages Scheduling for Data Redistribution between Clusters. In *Algorithms, models and tools for parallel computing on heterogeneous networks (HeteroPar'03) workshop of SIAM PPAM 2003, LNCS 3019*, pages 896–906, Czestochowa, Poland, September 2003.

[19] D. E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. Computing as a discipline. *Commun. ACM*, 32(1) :9–23, 1989.

[20] P. Crescenzi, D. Xiaotie, and C. H. Papadimitriou. On Approximating a Scheduling Problem. *Journal of Combinatorial Optimization*, 5 :287–297, 2001.

[21] D. Culler, R. Karp, R. Patterson, A. Sahay, K. E. Shauser, E. Santos, R. Subramonian, and T. Von Eiken. LogP : Toward a realistic model of parallel computation. In *4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, May 1993.

[22] The DAS-3 project : http://www.starplane.org/das3/.

[23] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel geometric algorithms for coarse grained multicomputers. *Proceedings of the ninth annual symposium on Computational geometry*, pages 298–307, 1993.

[24] Peter J. Denning. ACM President's Letter : What is experimental computer science ? *Commun. ACM*, 23(10) :543–544, 1980.

[25] Peter J. Denning. ACM president's letter : performance analysis : experimental computer science as its best. *Commun. ACM*, 24(11) :725–727, 1981.

[26] Peter J. Denning. Is computer science science ? *Commun. ACM*, 48(4) :27–31, 2005.

[27] F. Desprez and E. Jeannot. Improving the GridRPC Model with Data Persistence and Redistribution. In *3rd International Symposium on Parallel and Distributed Computing (ISPDC)*, Cork, Ireland, July 2004.

[28] DIET (Distributed Interactive Engineering Toolbox). http://www.ens-lyon.fr/~desprez/DIET/index.html.

[29] P.F. Dutot, L. Eyraud, G. Mounié, and D. Trystram. Bi-criteria Algorithm for Scheduling Jobs on Cluster Platforms. In *SPAA*, pages 125–132, 2004.

[30] Qemu : open source processor emulator http://www.qemu.org/.

[31] M. M. Eshagian and Y. C. Wu. Mapping heterogeneous task graphs onto heterogeneous system graphs. *Heterogeneous Computing Workshop (HCW'97)*, pages 147–160, April 1997.

[32] A. Esnard. Modèle pour la redistribution de donnés complexes. 16ièmes Rencontres francophones du parallélisme, RenPar'16, 2005.

[33] Dror G. Feitelson. Experimental Computer Science : The Need for a Cultural Change. Version sur internet : //www.cs.huji.ac.il/~feit/exp/related.html, December 2006.

[34] Ian Foster and Karl Kesselman, editors. *The GRID 2, Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004.

[35] Assefaw Hadish Hadish Gebremedhin, Jens Gustedt, Mohamed Essaïdi, and Isabelle Guérin Lassous andJan Arne Telle. PRO : A Model for the Design and Analysis of Efficient and Scalable Parallel Algorithms. *Nordic Journal of Computing*, 2006.

[36] A.V. Gerbessiotis and L.G. Valiant. Direct Bulk-Synchronous Parallel Algorithms. *Journal of Parallel and Distributed Computing*, 22(2) :251–267, 1994.

[37] C. Germain, V. Breton, P. Clarysse, Y. Gaudeau, T. Glatard, E. Jeannot, Y. Legré, C. Loomis, J. Montagnat, J.-M. Moureaux, A. Osorio, X. Pennec, and R. Texier. Grid-Enabling Medical Image Analysis. In *Third International Workshop on Biomedical Computations on the Grid (Bio-Grid 2005)*, Cardiff, UK, May 2005.

[38] T. F. Gonzalez and S. Sahni. Open Shop Scheduling to Minimize Finish Time. *J. ACM*, 23(4) :665–679, 1976.

[39] The Grid 5000 project : http://www.grid5000.org/.

[40] Grid explorer (GdX) : http://www.lri.fr/~fci/GdX/.

[41] Salim Hariri Haluk Topcuoglu and Min-You Wu. Task scheduling algorithms for heterogeneous processors. *8th IEEE Heterogeneous Computing Workshop (HCW'99)*, pages 3–14, April 1999.

[42] The HydroGrid Project. http://www-rocq.inria.fr/kern/HydroGrid/.

[43] E. Jeannot. Improving Middleware Performance with AdOC : an Adaptive Online Compression Library for Data Transfer. In *International Parallel and Distributed Processing Symposium 2005 (IPDPS'05)*, Denver, Colorado, USA, April 2005.

[44] E. Jeannot, B. Knutsson, and M Björkman. Adaptive Online Data Compression. In *IEEE High Performance Distributed Computing (HPDC'11)*, pages 379 – 388, Edinburgh, Scotland, July 2002.

[45] E. Jeannot and F. Wagner. Two Fast and Efficient Message Scheduling Algorithms for Data Redistribution through a Backbone. In *IEEE International Conference on Parallel and Distributed Processing Symposium (IPDPS)*, Santa-Fe, New-Mexico, USA, April 2004.

[46] E. Jeannot and F. Wagner. Messages Scheduling for Data Redistribution between Heterogeneous Clusters. In *IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2005)*, Phoenix, AZ, USA, November 2005.

[47] E. Jeannot and F. Wagner. Modeling, Predicting and Optimizing Redistribution between Clusters on Low Latency Networks. In *The First International Symposium on Frontiers in Networking with Applications (FINA 2006)*, Vienna, Austria, April 2006. IEEE.

[48] E. Jeannot and F. Wagner. Scheduling messages for data redistribution : an experimental study. *International Journal of High Performance Computing Applications*, 20(4) :443–454, November 2006.

[49] Y.-K. Kwok and I. Ahmad. Dynamic Critical-Path Scheduling : An Effective Technique for Allocating Task Graphs to Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 7(5) :506–521, May 1996.

[50] l. G. Valiant. A Briging Model for Parrallel Computation. *Communication of the ACM*, 33(8) :103–111, August 1990.

[51] A. Lastovetsky, X. Zuo, and P. Zhao. A Non-Intrusive and Incremental Approach to Enabling Direct Communications in RPC-based Grid Programming Systems. Technical Report UCD-CSI-2005-2,, University College Dublin, 2005.

[52] Arnaud Legrand, Loris Marchal, and Henri Casanova. Scheduling Distributed Applications : the SimGrid Simulation Framework. In *CCGRID*, pages 138–145, 2003.

[53] Paul Luckowicz, Walter F. Tichy, Ernst A. Heinz, and Lutz Prechelet. Experimental evaluation in computer science : a quantitative study. Technical Report iratr-1994-17, University of Karlshrue, Germany, 1994.

[54] M. Maheswaran, S. Ali, H.J. Siegel, D. Hengsen, and R.F. Freund. Dynamic Matching and Scheduling of a class of Independent Tasks onto Heterogeneous Computing System. In *Proceedings of the 8th Heterogeneous Computing Workshop (HCW '99)*, april 1999.

[55] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. A GridRPC Model and API for End-User Applications, December 2003. https://forge.gridforum.org/projects/gridrpc-wg/document/GridRPC_EndUser_16dec03/en/1.

[56] Padico : a Software Environment for Computational Grids. http://www.irisa.fr/paris/Padico/.

[57] Planet lab : http://www.planet-lab.org/.

[58] ProActive (Programming, Composing, Deploying on the GRID). http://www-sop.inria.fr/oasis/ProActive/.

[59] V. Sarkar. *Partitionning and Scheduling Parallel Program for Execution on Multiprocessors.* MIT Press, Cambridge MA, 1989.

[60] Anthony Sulistio, Chee Shin Yeo, and Rajkumar Buyya. A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. *Softw., Pract. Exper.*, 34(7) :653–673, 2004.

[61] Atsuko Takefusa, Satoshi Matsuoka, Hidemoto Nakada, Kento Aida, and Umpei Nagashima. Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms. In *HPDC*, 1999.

[62] iproute2+tc notes : http://snafu.freedom.org/linux2.2/iproute-notes.html.

[63] Walter F. Tichy. Should Computer Scientists Experiment More ? *Computer*, 31(5) :32–40, 1998.

[64] J. Turek, J. Wolf, and P. Yu. Approximate Algorithms Scheduling Parallelizable Tasks. In *Proceedings of the fourth annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'92)*, pages 323–332. ACM Press, 1992.

[65] F. Wagner. *Redistributrion de données à travers un réseau à haut débit.* PhD thesis, Université Henri Poincaré, 2005.

[66] R. Wolski, N. T. Spring, and J. Hayes. the network service : a distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6) :757–768, october 1999.

[67] The xen virtual machine monitor : http://www.cl.cam.ac.uk/research/srg/netos/xen/.

[68] Huaxia Xia, Holly Dail, Henri Casanova, and Andrew A. Chien. The MicroGrid : Using Online Simulation to Predict Application Performance in Diverse Grid Network Environments. In *CLADE*, page 52, 2004.

[69] T. Yang and A. Gerasoulis. DSC Scheduling Parallel Tasks on an Unbounded Number of Processors. *IEEE Transactions on Parallel and Distributed Systems*, 5(9) :951–967, September 1994.

[70] M.V. Zelkowitz and D.R. Wallace. Experimental models for validating technology. *Computer*, 31(5) :23–31, May 1998.

Annexe A

Curriculum Vitæ

Emmanuel Jeannot

Curriculum Vitæ Page 52

- Chargé de Recherche à l'INRIA (depuis sept. 2005).
- Maître de conférences à l'Université H. Poincaré – Nancy I (2000–2005).
- Postdoctorat INRIA dans l'ARC OURAGAN au Laboratoire Bordelais de Recherche en Informatique de l'Université de Bordeaux I (1999-2000).
- Thèse au Laboratoire de l'Informatique du Parallélisme, École Normale Supérieure de Lyon (1996-1999).
- Magistère et DEA d'informatique à L'ENS de Lyon.

Recherche Page 52

- **Au LORIA :** projet Résédas (2000-2002), équipe AlGorille (depuis 2002).
- **Au LaBRI :** postdoctorat, équipe alienor (1999-2000).
- **Au LIP :** thèse, projet ReMaP (1996-1999).
- **Thèmes de recherche :** ordonnancement (thèse, postdoc, LORIA), méta-computing et grilles informatiques (postdoc, LORIA), algorithmique hétérogène et distribuée (postdoc, LORIA), gestion des communications (LORIA), émulation de l'hétérogénéité (LORIA), parallélisme (thèse et postdoc), calcul numérique creux (postdoc), génération de code (thèse).
- **Collaborations internationales :** Jack Dongarra : Univ. tennessee ; Joël Goossens, Université Libre de Bruxelles.
- **Publications :** 12 revues internationales, 1 chapitre de livre, 24 conférences internationales, deux actes d'écoles thématiques.

Activités pédagogiques Page 60

- Maître de conférences à l'Université H. Poincaré, Nancy-1, IUT Nancy-Brabois, Département GTR de septembre 2000 à sept. 2005.
- **Domaines :** algorithmique parallèle et distribuée ; algorithmique et programmation en Pascal, C et java ; programmation des réseaux ; système d'exploitation ; base de données ; compilation ; graphisme.
- **Publics :** classe préparatoire *HEC* ; IUT ; DEUG MASS ; DEUG MIAS ; Magistère d'Informatique et Modélisation ; école d'ingénieur ; DEA

Activités administratives, responsabilités collectives Page 61

- Membre du comité de pilotage du RTP : "*calcul à haute performance et calcul réparti*"
- Responsable de GRID 5000 à Nancy.
- Membre du comité de programme de HCW'2004, Renpar(2003 et 2005), GRID 2005, IPDPS 2006, HPDC 2007, Europar 2007, IPDPS 2008.

Liste complète des publications Page 61

Curriculum Vitæ

État Civil

Date de naissance :	7 mai 1969
Lieu de Naissance :	Lyon 7ème
Nationalité :	française
Situation familiale :	marié, 3 enfants
Service national :	effectué (1994-95)
Situation actuelle :	Chargé de recherche 1ère classe, INRIA
Courriel :	`emmanuel.jeannot@loria.fr`

Cursus

Sept. 2006 – : CR1 à l'INRIA.

Sept. 2005 – sept. 2006 : Détaché à l'INRIA dans l'Unité de Recherche "*Lorraine*". Chercheur invité au laboratoire ICL de l'Université du Tennessee, Knoxville pendant 6 mois.

2000–2005 : Maître de conférences à l'université H. Poincaré, Nancy I.

1999–2000 : Postdoctorat INRIA : dans le cadre de l'ARC OURAGAN au LABRI, Bordeaux.

1996–1999 : Thèse d'informatique préparée et soutenue au Laboratoire de l'Informatique du Parallélisme à l'ENS de Lyon en tant qu'allocataire de recherche MENRT : "*Allocation de graphes de tâches paramétrés et génération de code*" sous la direction de Michel Cosnard. Soutenue le 8 octobre 1999 devant le jury composé de Yves Robert (président), Paul Feautrier et Jean-Claude König (rapporteurs), Michel Cosnard et Apostolos Gerasoulis.

1996 : DEA d'Informatique de Lyon mention bien et magistère d'informatique et modélisation de l'ENS Lyon, mention bien.

1992–1994 : Deux premières années du magistère d'informatique et modélisation de l'École Normale Supérieure de Lyon. Licence et maîtrise d'informatique de l'Université Claude Bernard-Lyon I, par équivalence.

1992 : DEUG des sciences de l'analyse et de la matière de l'Université Claude Bernard – Lyon I. Admis en magistère d'informatique et modélisation à L'École Normale Supérieure de Lyon.

Recherches

Une grande partie des travaux de recherches que j'ai menés porte sur le calcul parallèle et distribué. Les recherches dans ce domaine visent à exécuter efficacement un programme sur un ensemble de ressources dans le but soit d'accélérer le traitement, soit de travailler sur des données de très grande taille. Plus précisément, un des objectifs que je poursuis depuis mes débuts en recherche est de permettre une utilisation du parallélisme à la fois simple et efficace.

Thèse

Durant ma thèse, j'ai travaillé sur le parallélisme automatique et l'ordonnancement. Le parallélisme automatique consiste à transformer automatiquement un programme séquentiel en un programme parallèle. L'ordonnancement consiste à allouer dans le temps et dans l'espace des tâches sur des ressources distribuées.

Le graphe de tâches (GdT) est un modèle très utilisé pour la prédiction de performance et l'ordonnancement d'applications parallèles. Un algorithme d'ordonnancement statique prend en entrée un GdT et affecte, à chaque tâche, un processeur et une date de début d'exécution. Utiliser un GdT n'est pas une approche extensible (car pour les grandes valeurs des paramètres d'une application, le graphe de tâches correspondant peut ne pas tenir en mémoire) ni une approche adaptative (car un changement de machine cible ou des paramètres du programme impose de

recalculer l'ordonnancement).

Pour apporter une réponse à ces deux problèmes, j'ai étudié un modèle intermédiaire : le graphe de tâches paramétré (GTP). Un GTP est une représentation compacte et symbolique des graphes de tâches issus de certaines applications de calcul scientifique [42]. En outre, il peut facilement être déduit de certains programmes séquentiels comme des noyaux de calcul intensif présent dans les applications scientifiques.

L'objectif de ma thèse était donc, à partir d'un graphe de tâches paramétré, représentant un programme séquentiel, de générer un programme parallèle qui alloue efficacement les tâches sur les ressources.

Ma thèse se décompose en trois volets. (1) J'ai conçu un algorithme d'ordonnancement dynamique du GTP. Le coût mémoire de l'ordonnancement se trouve alors grandement réduit [12, 41]. (2) J'ai mis au point une heuristique d'allocation symbolique du GTP appelée SLC. Je garanti que cette allocation forme des grappes linéaires. Le temps et le coût mémoire de l'allocation sont alors indépendants de la valeur des paramètres [9, 11, 13, 39, 40]. (3) J'ai réalisé un prototype de générateur de code qui produit un programme multithreadé se conformant à l'allocation trouvée par SLC. J'obtiens ainsi un code parallèle portable et générique qui fonctionne pour toutes les valeurs des paramètres du programme [38].

Une partie des travaux de ma thèse s'est fait en collaboration avec Tao Yang [9, 13, 39, 40] de UCSB et Apostolos Gerasoulis de l'université de Rutgers.

Postdoctorat

Durant ma thèse, j'ai fait l'hypothèse que les ressources qui constituent l'infrastructure parallèle d'exécution étaient homogènes. C'est à dire que les nœuds de la machine parallèle sont identiques (même processeur, même quantité de mémoire, etc.).

À partir de mon postdoctorat je me suis intéressé aux environnements hétérogènes et distribués comme ceux constitués à partir de machines d'un laboratoire ou suite à une mise à jour partielle du matériel.

J'ai poursuivi d'octobre 1999 à septembre 2000, un postdoctorat dans le cadre de l'ARC INRIA *OURAGAN* (OUtils de Résolutions Appliqués aux Grands cAlculs Numériques), dirigée par Frédéric Desprez. J'ai travaillé au sein de l'équipe AliENor du LaBRI à Bordeaux.

L'action coopérative de recherche OURAGAN avait pour but de permettre l'utilisation, de manière aussi transparente que possible de Scilab, appelé "Scilab parallèle", sur une plateforme parallèle. Scilab est un logiciel scientifique dédié au calcul numérique (du style Matlab) et développé en grande partie à l'INRIA Rocquencourt. Cet environnement est utilisé dans le monde entier. J'ai travaillé autour de : (1) le calcul creux distribué. J'ai développé une interface entre Scilab et NetSolve, qui permet d'appeler à travers le réseau un ensemble de bibliothèques de calcul numérique creux. (2) l'allocation de tâches et équilibrage de charge. Il s'agit de réaliser et de mettre en œuvre des stratégies de placement statique de tâches (représentant des instructions Scilab$_{//}$) sur une architecture hétérogène et de réaliser l'équilibrage dynamique de la charge. Ce travail s'est déroulé dans le cadre d'un stage de DEA effectué par Bertrand Cirou et a donné lieu à une publication dans une conférence internationale [37]. (3) la persistance et la redistribution de données dans NetSolve. J'ai montré comment améliorer les performances des environnements pour le méta-computing en permettant aux serveurs de conserver leurs données localement et en pouvant redistribuer celles-ci directement [7, 31]. Les travaux sur Scilab$_{//}$ ont abouti à une publication dans une revue [10] et une conférence internationale [44].

Recherches au LORIA

Depuis ma nomination comme maître de conférences en septembre 2000, j'ai poursuivi mes travaux sur le calcul pour les plateformes distribuées et hétérogènes (grilles de calcul). J'ai

effectué mes recherches au sein du projet Résédas de l'INRIA (2000-2002), puis, à partir de 2002, dans l'équipe AlGorille dirigée par Jens Gustedt que nous avons créée ensemble. Au LORIA, mes recherches portent sur deux thèmes principaux :

- la *gestion des ressources* : l'ordonnancement d'applications sur la grille, la redistribution de données entre clusters, la compression adaptative
- les *environnements pour l'expérimentation* : l'émulation de l'hétérogénéité et la plateforme de recherche GRID'5000.

Le points commun de ces deux thèmes est l'algorithmique. Le premier thème concerne la modélisation et la conception d'algorithmes. Le deuxième thème concerne l'expérimentation des algorithmes.

Gestion des ressources

Le thème de la gestion des ressources consiste à mettre au point des services permettant une utilisation efficace des ressources distribuées et hétérogènes que j'ai commencé à étudier durant mon postdoctorat.

Un environnement (intergiciel ou middleware) permettant d'effectuer de manière transparente et efficace des calculs sur des ressources hétérogènes et distribuées doit reposer sur un ensemble de services gérants efficacement ces ressources. Les algorithmes que j'ai conçu visent à créer de nouveaux services ou à optimiser les services déjà présents dans les intergiciels proposés par la communauté. Les problèmes sur lesquels j'ai travaillé sont les suivants.

J'ai étendu mes travaux sur l'ordonnancement de tâches pour le cas où les ressources sont distribuées à une très grande échelle. Dans ce cas, un agent répartit les requêtes des utilisateurs (des clients) sur des ressources (des serveurs). C'est dans le contexte de ces recherches que se situe la thèse d'Yves Caniou que j'ai co-encadré. Plusieurs heuristiques ont été développées [35]. Elles sont basées sur un module de prédiction de performance non intrusif appelé le HTM (Historical Trace Manager), qui permet, en simulant la plateforme de déterminer la durée d'une tâche et l'utilisation des processeurs. Nous nous sommes rendu compte qu'une approche multicritères est nécessaire pour obtenir à la fois des performances et utiliser aux mieux les ressources disponibles. En effet, plusieurs acteurs agissent sur un tel environnements chacun ayant son propre objectif. Le client est intéressé par le temps de réponse de l'environnement. L'agent doit assurer que les ressources sont accédées équitablement par chaque utilisateur. Enfin les serveurs souhaitent que leurs ressources soient utilisées aux maximum. Dans [30, 32] nous avons proposées des heuristiques qui visent à apporter un compromis entre tous ses objectifs. Des expérimentations grandeur nature ont permis de valider l'approche proposée dans le modèle GridRPC [6].

Plus récemment, j'ai attaqué le problème de l'ordonnancement pour des ensembles de tâches (*workload*) stochastiques pour les grilles. En effet, la nature dynamique des applications et des utilisateurs nous force à abandonner les modèles où des suppositions fortes sont faites sur les tâches qui sont soumises (durée des tâches et date de soumission essentiellement) au profit de modèles où seule la structure de la soumission est connue et est déterminée par des lois de probabilité. L'algorithme est alors randomisé et les stratégies employées dépendent du type de lois qui définissent l'arrivée et la durée des tâches [5]. Les travaux en cours portent sur des problèmes de fiabilité [22] et de l'impact de la duplication lorsque les machines peuvent tomber en panne ou apparaître dynamiquement.

Le problème de la redistribution des données consiste à mettre au point des stratégies et des outils permettant à des données de transiter efficacement d'un cluster à un autre. Il s'agit d'un problème important dans le cadre de la distribution du calcul parallèle (comme le couplage de code). Ce problème fait suite aux travaux effectués durant mon postdoc sur la persistance et la redistribution dans le modèle GridRPC. D'un point de vue algorithmique le problème

étudié consiste à ordonnancer les messages de manière à ce que la bande passante agrégée par l'ensemble des messages ne dépasse pas, à tout instant, la bande passante du réseau qui relie les deux clusters. Dans ce contexte, j'ai co-encadré, avec Johanne Cohen, Nicolas Padoy pour un stage de première année de magistère de l'ENS de Lyon en juin et juillet 2002 [34]. J'ai aussi co-encadré la thèse de Frédéric Wagner sur ce thème. Nous avons proposé deux algorithmes $\frac{8}{3}$-approchés pour résoudre ce problème [4, 33]. Nous avons proposé une mèthode permettant de choisir le meilleur algorithme parmi plusieurs disponibles [3]. Plus récemment, nous avons résolus le problème pour le mode δ-port (chaque nœud peut faire plusieurs communications simultanément) [25]. Dernièrement nous avons travaillé sur l'utilisation des réseaux locaux pour équilibrer et accélérer la redistribution.

Une grille est parfois mise en place sur des réseaux relativement lents ce qui rend les communications extrêmement coûteuses. Pour palier à ce problème, nous avons développé une bibliothèque adaptative de compression à la volée de données (AdOC). La nature hétérogène et dynamique des grilles nécessite que les services qui y sont développés soient capables de s'adapter à l'environnement et à son évolution. En effet, une grille informatique peut être partagée par plusieurs utilisateurs et les performances des machines ou des réseaux qui la constitue peuvent varier très rapidement. Ainsi, en ce qui concerne AdOC, le niveau de compression des données transmises dépend de l'état du réseau et des ressources de calculs disponibles à chaque extrémité de la communication. L'adaptation du niveau de compression se fait en temps réel, tout au long du transfert des données. Par exemple, si le réseau est très rapide, alors il n'y a que très peu, voire pas de temps disponible pour compresser les données. Mais, si la bande passante décroît (à cause d'une congestion temporaire), AdOC le détecte et peut activer ou accroître la compression des données. Pour des raisons d'efficacité, AdOC recouvre le transfert des données avec la compression ou la décompression (une partie des données peut-être envoyée sur le réseau alors qu'une autre partie est en train d'être compressée). Pour ce travail, j'ai collaboré avec des chercheurs de l'Université de Mälardalens (Suède) et celle de Pennsylvanie [36]. Ces travaux ont abouti à la réalisation du logiciel AdOC [27]. AdOC est un cours d'intégration dans l'intergiciel NetSolve développé à l'Université du Tennessee.

Environnements pour l'expérimentation

Il existe plusieurs manières de prouver l'efficacité des algorithmes que l'on met au point. Tout d'abord, on peut faire des preuves sur les caractéristiques de l'algorithme (complexité, ratio d'approximation, etc.). Cependant, la complexité des environnements de calcul distribué et des grilles peut rendre la modélisation difficile et la mise en évidence de caractéristiques formelles presque impossible. Une approche expérimentale est donc indispensable dans ce cas. En outre, l'expérience permet de valider les modèles utilisés. Dans ce contexte, l'informatique est une science expérimentale et donc l'expérience un sujet d'étude scientifique. Mettre au point des environnements pour conduire des expériences est un des défis qu'il convient de relever. Dans ce domaine, notre contribution est double.

Tout d'abord, nous avons travaillé sur l'émulation de l'hétérogénéité. Le problème de l'émulation de l'hétérogénéité consiste à mettre au point ou à utiliser des outils systèmes pour transformer un cluster de machines homogènes en un cluster hétérogène. L'émulation de l'hétérogénéité est nécessaire pour tester des algorithmes dédiés à l'hétérogénéité dans un cadre expérimental totalement contrôlé et reproductible. Pour aboutir à cette émulation, il faut être capable de dégrader de manière individuelle les caractéristiques des nœuds (Vitesse, mémoire, bande passante du réseau, latence). Ce travail a fait l'objet d'un stage pendant l'été 2004 et d'un stage durant l'été 2005. Nous avons testé notre outil sur GridExplorer et montré que nous sommes capables de dégrader indépendamment les différentes caractéristiques d'un nœud [23].

Enfin, je participe au projet grid'5000 [26]. L'objectif de GRID 5000 est de fournir un instrument pour réaliser des expériences sur les grilles informatiques. Il s'agit de mettre en place des clusters de machines sur des sites distribuées dans toute la France. À terme GRID 5000 devra regrouper 5000 processeurs répartis sur une dizaine de sites ou laboratoires français et permettra de tester les algorithmes, les protocoles, les services et les environnements développés par la communauté. À ce jour, 9 sites ont été sélectionnés (Lille, Rennes, Orsay, Nancy, Bordeaux, Lyon, Grenoble, Toulouse et Nice). Ce projet, par sa dimension et son approche (expérimentale), est unique au monde. À mon initiative, Nancy est devenu le $9^{\text{ème}}$ nœud de grid'5000.

Participations à des projets

- **Eureka EuroTOPS :** Participation aux tâches PlusPyr (1996) ordonnancement de graphes de tâches paramétrés (1997-98).
- **NSF/CNRS :** LIP-ENS Lyon, Univ. Rutgers, Univ. Santa-Barbarra (1996-1999). Collaboration avec Tao Yang (UCSB) et Apostolos Gerasoulis (Univ. Rutgers) sur l'ordonnancement de graphes de tâches paramétrés.
- **ARC INRIA OURAGAN :** Projet Métalau, INRIA Rocquencourt ; Projet ReMaP, LIP ENS Lyon, INRIA Rhône-Alpes ; Projet Résédas, LORIA, INRIA Lorraine ; équipe ALiENor, LaBRI, Université Bordeaux I et ENSERB ; équipe PaLaDIN, LaRIA, Université de Picardie Jules Verne ; équipe SRDP, LIFC, Université de Franche Comté. (1998–2000). Postdoctorant (1999–2000) participation à la réalisation de Scilab$_{//}$, travaux sur la persistance et la redistribution des données.
- **ACI GRID ASP :** Projet ReMap (INRIA) ; projet Résédas (INRIA) ; équipe SRDP du LIFC ; Département de Physique des Matériaux, UMR 5586, Université de Lyon 1 ; Laboratoire de Physique, ENS Lyon, Laboratoire d'Analyse Numérique (LAN), UMR 5585, Université de Lyon 1 ; UMR 7565 CNRS-UHP-INPL, Structure et Réactivité des Systèmes Moléculaires Complexes (SRSMC) Université Henri Poincaré - Nancy I ; laboratoire des Sciences de la Terre (LST), ENS Lyon (2001-2004). Action Concertée Incitative pluridisciplinaire. Responsable de la partie Exploration de surfaces d'énergie potentielle en collaboration avec le SRSMC [28].
- **RNTL GASP :** Sun Labs Meylan, projet ReMaP, projet Résédas, laboratoire IRCOM, équipe SRDP du LIFC, laboratoire des sciences de la terre ENS Lyon(2001-2004). Participation à la conception de DIET, un environnement de calcul distribué développé dans le projet Graal de l'INRIA Rhône-Alpes.
- **ARC INRIA RedGRID :** Projet ReMaP, Projet Paris, projet ScAlApplix, équipe AlGorille (2003 – 2004) : mise au point d'algorithmes et de bibliothèques de redistribution de données sur la grille.
- **NSF/INRIA :** entre Univ. du Tennessee, Projet INRIA ReMaP et l'équipe AlGorille. Portage de AdOC dans NetSolve ; ordonnancement dans le modèle GridRPC ; prédiction de performance des routines d'algèbre linéaire ; ordonnancement fiable et robuste.
- **ACI Masse de données Grid-Explorer :** dirigée par Franck Cappello (15 laboratoires en France). Mise au point du module d'émulation de l'hétérogénéité wrekavoc (2003-2006).
- **ACI Masse de données AGIR :** (2004-2007) LAL, LRI, CAL (Centre Antoine Lacassagne), LPC, LORIA, CREATIS, I3S, CRAN, LIMSI, INRIA Sophia-Antipolis. Réalisation d'un environnement interactif pour l'imagerie médicale [8, 29]. Responsable de la partie transfert avec compression et de la partie middleware.

- **Réseaux d'excellence CoreGRID :** Participation au working group gestion des ressources/ordonnancement du réseaux d'excellence CoreGRID du 6ème programme cadre de l'Union Européenne (42 partenaires). Responsable de la partie benchmark pour le partenaire CNRS.

Développement de logiciels

Mes travaux ont donné lieu à de nombreux développements de logiciels :

- PlusPyr : logiciel d'extraction et d'exécution de graphes de tâches paramétrés sur machine parallèle. Déposé à l'APP. En collaboration avec M. Cosnard, M. Loi et L. Rougeot. Extension du programme en intégrant des heuristiques d'ordonnancement stsatique.
- Participation à la Polylib : bibliothèque de calcul sur les polyèdres, disponible sur http://icps.u-strasbg.fr/PolyLib/. Écriture de la fonction d'évaluation d'un polynôme de Ehrhart. En collaboration avec V. Loechner.
- AdOC (Adaptive Online Compression) : bibliothèque de compression dynamique et adaptative pour le transfert de données. Déposé à l'APP disponible sur http://www.loria.fr/~ejeannot/adoc sous licence LGPL. 4500 lignes de code (2 ans de développement).
- Wrekavoc. Outil d'émulation de l'hétérogénéité. Il s'agit de transformer un cluster homogène en cluster hétérigè,ne pourt pouvoir expérimenter des algorithmes conçus pour ces environnements. Développé dans la cadre de l'ACI grid GDX en collaboration avec Marc Thierry, Louis-Caude Canon, et Olivier Dubuisson (6000 lignes de code) disponible sur le gforge de l'INRIA : http://wrekavoc.gforge.inria.fr

Encadrement de jeunes chercheurs

Louis-Claude Canon (master et thèse). J'ai encadré le stage de Master de Louis-Claude Canon sur l'analyse multicritère de méta-heuristiques pour l'ordonnancement d'applications sur plates-formes distribuées à large échelle. L'idée est d'utiliser des méta-heuristiques, comme les algorithmes génétiques, pour trouver des bornes inférieures à des problèmes d'optimisation multi-critère et de les comparer à des heuristiques. L'objectif est de construire des heuristiques trouvant des solutions proches des méta-heuristiques pour un coût algorithmique moindre. Depuis septembre 2007, Louis-Claude poursuivi en thèse sous ma direction pour le sujet : *Gestion fiable de la dynamicitée des environnements parallèles distribuées.*

Frédéric Wagner (thèse). De décembre 2002 à décembre 2005, j'ai co-encadré, à 80% la thèse de Frédéric Wagner (Bourse Région/INRIA-Lorraine) sur les problèmes de redistribution de données sur les grilles. Le but était de mettre au point des algorithmes de redistribution entre grappes de PC reliées par un backbone. Ces travaux ont donné lieu à une publication dans IPDPS 2004, RENPAR 2005 et PDCS 2005 ("*best paper award*"), FINA'2006 et à un article dans la revue IEEE Transaction on Parallel and Distributed Systems. Frédéric Wagner a soutenu sa thèse le 14 décembre 2005. Il est actuellement maître de conférences à l'ENSIMAG de Grenoble.

Yves Caniou (thèse). Entre octobre 2001 et décembre 2004, j'ai co-encadré à 80% la thèse d'Yves Caniou (Bourse Région-INRIA Lorraine) sur les problèmes d'ordonnancement d'applications pour les plateformes de méta-computing. Le but était de réaliser et de mettre en œuvre des algorithmes de placement de calculs sur des ressources distribuées hétérogènes. Ces travaux ont donné lieu à une publication dans RENPAR 2002 et 2004, dans HCW 2003, dans un workshop d'ICPADS 2004 et Europar 2004. Un article dans la revue International Journal of High Performancve Computing and Applications est accepté. Yves

Caniou a soutenu sa thèse le 16 décembre 2004. Il est actuellement maître de conférences à l'Université de Lyon I.

Bertrand Cirou (DEA). J'ai encadré en 2000 le stage de DEA de Bertrand Cirou sur le thème de l'ordonnancement de graphes de tâches en milieu hétérogène. Il a mis au point un algorithme d'ordonnancement appelé *triplet* en se basant sur les techniques de regroupement (*clustering*) qui sont utilisées dans le cas de ressources homogènes. Ces travaux ont donné lieu à une publication [37] .B. Cirou a effectué une thèse au LaBRI.

Laurent Bobelin (DEA). J'ai co-encadré en 1999 le stage de DEA de Laurent Bobelin sur des heuristiques de placement de données pour des algorithmes à parallélisme mixte au LIP à l'ENS de Lyon.

Encadrement d'ingénieurs et de stagiaires

Eloi Dubois (stage de licence). AdOC : extension à la compression avec perte d'images médicales (2 mois, 2007).

Olivier Dubuisson (stage ingénieur CNAM), Wrekavoc : extension et tests à large échelle. Mise en place d'une émulation de réseau réaliste entre les ilôts (un an en 2007).

Xavier Delaruelle (Ingénieur associé INRIA) Mise en place et maintenance du cluster GRID 5000 pour le site de Nancy (2 ans à partir de Sept. 2005)

Louis-Claude Canon (Stage 1A ESEO) Test et amélioration de Wrekavoc, algorithmes de redistribution (3 mois, 2005).

Marc Thierry (Stage supelec 2A) Wrekavoc : outil pour la gestion de l'hétérogénéité des clusters (2 mois, 2004).

Hanane Moustain Billah (Initiation à la recherche) portage de AdOC sur Windows (1 mois, 2004).

Ndoli-Guillaume Assielou, Bertrand Benoit et Alexandre Dombrat (Initiation à la recherche) tests d'algorithmes rapides pour AdOC (1 mois, 2003).

Karen Montemont (Maitrise) Site web pour l'école Grid 2002 et pour le logiciel AdOC (1 mois, 2002).

Cinq publications les plus significatives

- M. Cosnard, E. Jeannot et T. Yang. Compact DAG Representation and Its Symbolic Scheduling. *Journal of Parallel and Distributed Computing*, 64(8) :921–935, août 2004.
- Y. Caniou and E. Jeannot, Multi-Criteria Scheduling Heuristics for GridRPC Systems, In *International Journal of High Performance Computing Applications* 20(1) : 61–76, spring 2006.
- V. Berten, J. Goossens et E. Jeannot, On the Distribution of Sequential Jobs in Random Brokering for Heterogeneous Computational Grids. IEEE Transaction on Parallel and Distributed Systems 17(2) :113–124, février 2006.
- J. Cohen, E. Jeannot, N. Padoy, et F. Wagner. Message Scheduling for Parallel Data Redistribution between Clusters. IEEE Transaction on Parallel and Distributed Systems 17(10) : 1163-1175, novembre 2006.
- E. Jeannot. Improving Middleware Performance with AdOC : an Adaptive Online Compression Library for Data Transfer. In *International Parallel and Distributed Processing Symposium 2005 (IPDPS'05).*

Prix/recompenses

L'article *Messages Scheduling for Data Redistribution between Heterogeneous Clusters* [25] à reçu le "*Best Paper Award in Algorithms*" à la 17$^{\text{th}}$ IASTED International Conference on Parallel and Distributed Computing Systems – PDCS 2005. Phoenix AZ, 14-16 novembre 2005.

Séjours à l'étranger et invitations

De janvier 2006 à juin 2006 j'ai passé 6 mois à l'Université du Tennessee dans le *Innovative Computing Laboratory* (ICL) dirigé par Jack Dongarra. Durant cette période, j'ai travaillé à la modélisation e la factorisation LU [43], à l'ordonnancement dans GridSolve [1] et à l'ordonnancement robuste de graphes de tâches [20].

J'ai aussi été invité à présenter mes travaux au niveau international à ces différentes réunions :

- *Multi-criteria Scheduling for Heterogeneous and Distributed Systems*, Workshop on Scheduling for large scale distributed platforms, août 2004, Aussois.
- *Improving Grid Middleware with Adaptive Online Compression (AdOC)*, Future Generation Grids - FGG 2004, novembre 2004, Dagstuhl, Germany.
- *Adaptive Online Compression (AdOC) and Scheduling*, French-Japan Grid Workshop at NII décembre 2004, Tokyo, Japan.
- *Ordonnancement de messages pour la redistribution de données entre grappes d'ordinateur*, séminaire à l'Université Libre de Bruxelles, février 2005, Bruxelles, Belgique.
- *Messages Scheduling for Data Redistribution (An experimental study)*, 2nd workshop on Scheduling for large-scale distributed platforms, novembre 2005, San-Diego, USA.
- *Improved Scheduling Strategies for Agent-Client-Server Middleware*, workhop on Clusters and Computational Grids for Scientific Computing, septembre 2006, Flat Rock, USA.
- *Bi-objective Scheduling Algorithms for Optimizing Makespan and Reliability on Heterogeneous Systems and some word about experimental environments (GRID'5000 and Wrekavoc)* , séminaire à l'Université du Minnesota, mars 2006, Minneapolis, USA.
- *Fast and Efficient Total Exchange on Two Clusters*, séminaire à University College Dublin, mars 2006, Dublin, Ireland.
- *Modeling the LU Factorization on SMP clusters*, présentation invitée au *Meeting on Optimization of Parallel Routines and Applications*, Murcia, Espagne, Juin 2007.

Activités pédagogiques

Enseignements en tant que vacataire

De 1995 à 2000, j'ai effectué des enseignements sous le statut de vacataire. Les principaux thèmes abordés sont : la programmation, l'algorithmique, la compilation et le parallélisme.

J'ai touché un large public, allant des classes préparatoires HEC, aux troisième années d'école d'ingénieurs (ENSERB) ainsi qu'aux deuxièmes années de magistère d'informatique et modélisation (MIM) à l'ENS de Lyon, en passant par les premiers cycles universitaires (DEUG, IUT, de l'Université Claude Bernard – Lyon I). Le nombre total d'heures effectuées atteint 376 heures.

Le tableau suivant résume mes activités d'enseignement :

Sujet	**Formation**	**Type et durée**
Algorithmique et programmation en Pascal	Prépa HEC, Lycée Ampère	TD ; 116 heures
	$2^{\text{ème}}$ année IUT de chimie, UCBL	TD ; 56 heures
	$1^{\text{ère}}$ année DEUG MASS, UCBL	TP ; 32 heures
Algorithmique et programmation en C	$2^{\text{ème}}$ année DEUG MIAS, UCBL	TP ; 48 heures
Cryptographie à clé public par la méthode du sac à dos	$1^{\text{ère}}$ année MIM à l'ENS Lyon	Projet de C 16 heures
Compilation du *LaX* en assembleur *SPARC*	$2^{\text{ème}}$ année de MIM à l'ENS Lyon	Projet de compilation 32 heures
Algorithmique parallèle et MPI	$2^{\text{ème}}$ année de MIM à ENS Lyon	TD ; 48 heures
Algorithmique parallèle et PM2 : "le solitaire"	$3^{\text{ème}}$ année de l'ENSERB	Module pratique 12 heures
Graphisme interactif	$2^{\text{ème}}$ année de l'ENSERB	TD ; 16 heures

Enseignements en tant que maître de conférences

J'ai enseigné à l'IUT Nancy Brabois de l'Université Henri Poincaré (UHP) de Nancy au département Génie des Télécommunications et des Réseaux (GTR) ainsi qu'en DEA d'Informatique de septembre 2000 à juin 2005. Au sein du département, j'ai assumé quatre enseignements différents : TD réseaux, algorithmique et programmation, base de données et systèmes d'exploitation. Le tableau suivant résume mes activités d'enseignement à l'UHP.

Sujet	**Année**	**durée**	**Période**
Algorithmique et programmation en C/Java	$1^{\text{ère}}$	TD et TP : 70 heures	2000-2003 ; 2004-2005
TD réseaux	$1^{\text{ère}}$	TD : 80 heures	2000-2003
Cours système, UNIX	$2^{\text{ème}}$	CM : 15 heures	2000-2004
TD système, UNIX	$2^{\text{ème}}$	TD : 40 heures	2000-2005
TD base de données	$2^{\text{ème}}$	TD-TP : 46 heures	2002-2005
Cours base de données	$2^{\text{ème}}$	CM : 10 heures	2003-2005
Cours programmation en C	$2^{\text{ème}}$	CM : 10 heures	2003-2004
Programmation en C	$2^{\text{ème}}$	TD et TP : 40 heures	2003-2005
algorithmique parallèle et distribuée	DEA	CM : 10 heures	2003-2005

Responsabilités pédagogiques

J'ai été responsable des projets tutorés du département GTR ainsi que de l'enseignement en informatique des deux années. J'ai participé à la rédaction du nouveau programme pédagogique national des départements GTR de France.

Activités administratives, responsabilités collectives

- Responsable du projet Grid 5000 à Nancy (Achat du cluster, mise en service et maintenance du cluster, organisation du site, politique locale). Membre du comité de pilotage national : définition de la stratégie est des actions à mener.
- Membre du comité de pilotage du réseau thématique pluridisciplinaire (RTP) "*Calcul à hautes performances et calcul réparti*" du département STIC du CNRS, animé par Brigitte Plateau et Yves Robert.
- Organisateur de l'école thématique GRID'2002 [15], sur le thème du méta-computing qui s'est tenue à Aussois (73) du 2 au 6 décembre 2002. Elle a regroupé plus de 130 chercheurs et ingénieurs des disciplines (informatique, physique, chimie, etc...) travaillant sur le thème de la grille.
- Coorganisateur avec Stéphane Vialle et Jens Gustedt de l'école GridUSe [14] (Metz, juin 2004) sur l'utilisation des services dans les grilles (60 participants).
- Membre du comité de programme de IEEE Heterogeneous Computing Workshop (HCW) 2004, de RenPar (depuis 2003), de IEEE Grid 2005, de IEEE International Parallel & Distributed Processing Symposium (IPDPS) 2006 et 2008, de High Performance Distributed Computing (HPDC) 2007, Europar 2007, Heteropar 2007, HCW 2008, CCGrid 2008.
- Relecteur de nombreuses revues dont : IEEE Trans. on Parallel and Distributed Systems, Parallel Processing Letters, Parallel Computing, Information Processing Letters, Theoretical Computer Science, Journal of Parallel and Distributed Computing, Computing and Informatics, etc. Ainsi que des conférences suivantes : IPPS'98, IPPS'2000, Europar (1998,99,00,02,05,07) ICPP'2000, PaCT'98, SPAA'01, PACT'01, STACS'02, HCW'04, STACS'05, IPDPS 2006, HPDC 2006, HCW 2007, HPDC 2007, Europar 2007, etc.
- Membre de la commission de spécialistes 27ème section de l'Université H. Poincaré. 2002-2004 : suppléant ; 2004 à 2005 : titulaire.
- Membre du comité de rédaction de *la lettre du LORIA* (2002-2005).
- Membre de la commission de recrutement des chercheurs contractuels (2004-2005) et des ingénieurs (2006 à ce jour) du LORIA.

Liste complète des publications

Journaux internationaux

[1] Emmanuel Jeannot, Keith Seymour, Asym Yarkhan, and Jack J. Dongarra. Improved Runtime and Transfer Time Prediction Mechanisms in a Network Enabled Servers Middleware. *Parallel Processing Letters*, 17(1) :47–59, March 2007.

[2] Raphël Bolze, Franck Cappello, Eddy Caron, Michel Daydé, Frédéric Desprez, Emmanuel Jeannot, Yvon Jégou, Stephane Lanteri, Julien Leduc, Noredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, Benjamin Quetier, Olivier Richard, El-Ghazali Talbi, and Iréa Touche. Grid'5000 : A Large Scale And Highly Reconfigurable Experimental Grid Testbed. *International Journal of High Performance Computing Applications*, 20(4) :481–494, November 2006.

[3] E. Jeannot and F. Wagner. Scheduling messages for data redistribution : an experimental study. *International Journal of High Performance Computing Applications*, 20(4) :443–454, November 2006.

[4] J. Cohen, E. Jeannot, N. Padoy, and F. Wagner. Message Scheduling for Parallel Data Redistribution between Clusters. *IEEE Transactions on Parallel and Distributed Systems*, 17(10) :1163–1175, October 2006.

[5] V. Berten, J. Goossens, and E. Jeannot. On the Distribution of Sequential Jobs in Random Brokering For Heterogeneous Computational Grids. *IEEE Transactions on Parallel and Distributed Systems*, 17(2) :113–124, 2006.

[6] Y. Caniou and E. Jeannot. Multi-Criteria Scheduling Heuristics for GridRPC Systems. *International Journal of High Performance Computing Applications*, 20(1) :61–76, spring 2006.

[7] E. Caron, B. Del-Fabbro, F. Desprez, E. Jeannot, and J.-M. Nicod. Managing data persistence in network enabled servers. *Scientific Programming Journal*, 13(4) :333–354, 2005.

[8] C. Germain, V. Breton, P. Clarysse, Y. Gaudeau, T. Glatard, E. Jeannot, Y. Legré, C. Loomis, I. Magnin, J. Montagnat, J.-M. Moureaux, A. Osorio, X. Pennec, and R. Texier. Grid-Enabling Medical Image Analysis. *Journal of Clinical Monitoring and Computing*, 19(4–5) :339–349, October 2005.

[9] Michel Cosnard, Emmanuel Jeannot, and Tao Yang. Compact Dag Representation and its Symbolic Scheduling. *Journal of Parallel and Distributed Computing*, 64(8) :921 – 935, August 2004.

[10] E. Caron, S. Chaumette, S. Contassot-Vivier, F. Desprez, E. Fleury, C. Gomez, M. Goursat, E. Jeannot, D. Lazure, F. Lombard, J.M. Nicod, L. Philippe, M. Quinson, P. Ramet, J. Roman, F. Rubi, S. Steer, F. Suter, and G. Utard. Scilab to Scilab$_{//}$, the OURAGAN Project. *Parallel Computing*, 27(11), 2001.

[11] M. Cosnard and E. Jeannot. Automatic Parallelization Techniques Based on Compact DAG Extraction and Symbolic Scheduling. *Parallel Processing Letters*, 11(1) :151–168, 2001.

[12] M. Cosnard and E. Jeannot. Compact DAG Representation and Its Dynamic Scheduling. *Journal of Parallel and Distributed Computing*, 58(3) :487–514, September 1999.

Chapitre de livre

[13] Leonidas S. Pitsoulis and Panos M. Pardalos, editors. *Nonlinear Assignment Problems : Algorithms and Applications*, chapter Symbolic Scheduling of Parameterized Task Graphs on Parallel Machines. Kluwer Academic Publishers, November 2000. ISBN 0-7923-6646-8 (with T. Yang and M. Cosnard).

Ouvrages collectifs édités en français

[14] S. Vialle, J. Gustedt, and E. Jeannot, editors. *GridUSe-2004 : Ecole thématique sur la Globalisation des Ressources Informatiques et des Données : Utilisation et Services.* Supelec, June 2004.

[15] J. Gustedt, E. Jeannot, J.-L. Pazat, and S. Vialle, editors. *École GRID 2002.* INRIA, December 2002. École thématique sur la globalisation des ressources et des données, Aussois, France.

Conférences internationnales avec publication des actes et comité de lecture

[16] L.-C. Canon and E. Jeannot. A Comparison of Robustness Metrics for Scheduling DAGs on Heterogeneous Systems. In *Sixth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (HeteroPar'07), in conjounction with The 2007 IEEE International Conference on Cluster Computing (cluster 2007)*, pages 568–567, Austin, Texas, September 2007.

[17] E. Jeannot and Luiz-Angelo Steffenel. Fast and Efficient Total Exchange on Two Clusters. In *the 13th International Euro-Par Conference*, volume 4641 of *LNCS*, pages 848–857, Rennes, France, August 2007. Springer Verlag.

[18] Luiz-Angelo Steffenel and E. Jeannot. Total Exchange Performance Prediction on Grid Environments : modeling and algorithmic issues. In *the CoreGRID Symposium (CoreGRID'07)*, pages 131–140, Rennes, France, August 2007.

[19] Jack J. Dongarra, Emmanuel Jeannot, Erik Saule, and Zhiao Shi. Bi-objective Scheduling Algorithms for Optimizing Makespan and Reliability on Heterogeneous Systems. In *19th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'07)*, San Diego, CA, USA, June 2007.

[20] Z. Shi, E. Jeannot, and J. J. Dongarra. Robust Task Scheduling in Non-Deterministic Heterogeneous Systems. In *Proceedings of IEEE International Conference on Cluster Computing*, Barcelona, Spain, October 2006. IEEE.

[21] E. Jeannot and F. Vernier. A Practical Approach of Diffusion Load Balancing Algorithm. In *12th International Euro-Par Conference*, volume 4128 of *LNCS*, pages 211–221, Dresden, Germany, August 2006. Springer Verlag.

[22] V. Berten, J. Goossens, and E. Jeannot. A Probabilistic Approach for Fault Tolerant Multiprocessor Real-time Scheduling. In *14th Workshop on Parallel and Distributed Real-Time Systems*, Island of Rhodes, Greece, April 2006.

[23] L.-C. Canon and E. Jeannot. Wrekavoc a Tool for Emulating Heterogeneity. In *15th IEEE Heterogeneous Computing Workshop (HCW'06)*, Island of Rhodes, Greece, April 2006.

[24] E. Jeannot and F. Wagner. Modeling, Predicting and Optimizing Redistribution between Clusters on Low Latency Networks. In *The First International Symposium on Frontiers in Networking with Applications (FINA 2006)*, Vienna, Austria, April 2006. IEEE.

[25] E. Jeannot and F. Wagner. Messages Scheduling for Data Redistribution between Heterogeneous Clusters. In *IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2005)*, Phoenix, AZ, USA, November 2005.

[26] Franck Cappello, Eddy Caron, Michel Dayde, Frédéric Desprez, Yvon Jegou, Pascale Primet, Emmanuel Jeannot, Stephane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Benjamin Quetier, and Olivier Richard. Grid'5000 : a large scale, reconfigurable, controlable and monitorable Grid platform. In *6th IEEE/ACM International Workshop on Grid Computing (GRID 2005)*, pages 99–106, Seattle, WA, USA, November 2005.

[27] E. Jeannot. Improving Middleware Performance with AdOC : an Adaptive Online Compression Library for Data Transfer. In *International Parallel and Distributed Processing Symposium 2005 (IPDPS'05)*, Denver, Colorado, USA, April 2005.

[28] E. Jeannot and G. Monard. Computing Molecular Potential Energy Surface with DIET. In *International Conference on Information Technology (ITCC2005)*, pages 286 – 291, Las-Vegas, Nevada, USA, April 2005.

[29] C. Germain, V. Breton, P. Clarysse, Y. Gaudeau, T. Glatard, E. Jeannot, Y. Legré,

C. Loomis, J. Montagnat, J.-M. Moureaux, A. Osorio, X. Pennec, and R. Texier. Grid-Enabling Medical Image Analysis. In *Third International Workshop on Biomedical Computations on the Grid (Bio-Grid 2005)*, Cardiff, UK, May 2005.

[30] Y. Caniou and E. Jeannot. Experimental Study of Multi-Criteria Scheduling Heuristics for GridRPC Systems. In *ACM-IFIP Euro-Par*, Pisa, Italy, September 2004.

[31] F. Desprez and E. Jeannot. Improving the GridRPC Model with Data Persistence and Redistribution. In *3rd International Symposium on Parallel and Distributed Computing (ISPDC)*, Cork, Ireland, July 2004.

[32] Y. Caniou and E. Jeannot. Efficient Scheduling Heuristics for GridRPC Systems. In *QOS and Dynamic System workshop of IEEE ICPADS (International Conference on Parallel and Distributed Systems) conference*, pages 621 – 630, New-Port Beach, California, July 2004.

[33] E. Jeannot and F. Wagner. Two Fast and Efficient Message Scheduling Algorithms for Data Redistribution through a Backbone. In *IEEE International Conference on Parallel and Distributed Processing Symposium (IPDPS)*, Santa-Fe, New-Mexico, USA, April 2004.

[34] Johanne Cohen, Emmanuel Jeannot, and Nicolas Padoy. Messages Scheduling for Data Redistribution between Clusters. In *Algorithms, models and tools for parallel computing on heterogeneous networks (HeteroPar'03) workshop of SIAM PPAM 2003, LNCS 3019*, pages 896–906, Czestochowa, Poland, September 2003.

[35] Y. Caniou and E. Jeannot. New Scheduling Heuristics in the Client-Agent-Server Model. In *IEEE Heterogeneous Computing Workshop (HCW'03)*, Nice, France, April 2003.

[36] E. Jeannot, B. Knutsson, and M Björkman. Adaptive Online Data Compression. In *IEEE High Performance Distributed Computing (HPDC'11)*, pages 379 – 388, Edinburgh, Scotland, July 2002.

[37] B. Cirou and E. Jeannot. Triplet : a Clustering Scheduling Algorithm for Heterogeneous Systems. In *IEEE ICPP International Workshop on Metacomputing Systems and Applications (MSA'2001)*, Valencia, Spain, September 2001.

[38] E. Jeannot. Automatic multithreaded parallel program generation for message passing multiprocessors using parameterized task graphs. In *International Conference 'Parallel Computing 2001' (ParCo2001)*, Naples, Italy, September 2001.

[39] M. Cosnard, E. Jeannot, and T. Yang. SLC : Symbolic Scheduling for Executing Parameterized Task Graphs on Multiprocessors. In *International Conference on Parallel Processing (ICPP'99)*, Aizu Wakamatsu, Japan, September 1999.

[40] M. Cosnard, E. Jeannot, and T. Yang. Symbolic Partitionning and Scheduling of Parameterized Task Graphs. In *IEEE International Conference on Parallel and Distributed Systems (ICPADS'98)*, Tainan, Taiwan, December 1998.

[41] M. Cosnard, E. Jeannot, and L. Rougeot. Low Memory Cost Dynamic Scheduling of Large Coarse Grain Task Graphs. In *IEEE International Parallel Processing Symposium (IPPS'98)*, Orlando, Florida, April 1998. IEEE.

[42] M. Cosnard and E. Jeannot. Automatic Coarse-Grained Parallelization Techniques. In Grandinetti and Kowalik, editor, *NATO workshop : Advances in High Performance Computing*. Kluwer academic Publishers, 1997.

Conférence internationale sans publication des actes, avec comité de lecture

[43] Jack Dongarra, Emmanuel Jeannot, and Julien Langou. Modeling the LU Factorization

for SMP Clusters. In *4th International Workshop on Parallel Matrix Algorithms and Applications (PMAA'06)*, Rennes, France, September 2006.

[44] F. Desprez, E. Fleury, E. Jeannot, F. Suter, and J-M. Nicod. Computational servers in a metacomputing environment. In *SIAM International Workshop on Parallel Matrix Algorithms and Applications*, Neuchâtel, Switzerland, August 2000.

Autres Conférences avec publication des actes et comité de lecture

[45] Y. Caniou and E. Jeannot. Le HTM : un module de prédiction de performance non-intrusif pour l'ordonnancement de tâches sur plate-forme de meta-computing. In *16 ème Rencontres Francophones du Parallèlisme (RENPAR 2005)*, Le Croisic, France, April 2005.

[46] Y. Caniou and E. Jeannot. Ordonnacemement pour la grille : une extension de MCT. In *14 ème Rencontres Francophones du Parallèlisme (RENPAR 2002)*, Hammamet, Tunisie, April 2002.

Colloques internationaux sans comité de selection

[47] M. Cosnard and E. Jeannot. On the Efficiency of Dynamic Scheduling for Automatic Parallelization. In *Workshop : Scheduling in Computer and Manufacturing Systems*, Dagstuhl, Germany, June 1997.

[48] M. Cosnard and E. Jeannot. Building and Scheduling Coarse Grain Task Graphs. In *Workshop on Scheduling in Parallel and Distributed Systems (WSPDS'96)*, CIRM, Marseille, France, June 1996.

Colloques francophones sans comité de selection

[49] Y. Caniou and E. Jeannot. Limitation des études validées par Simulation. In *Ecole thématique sur la Globalisation des Ressources Informatique et des Données : Utilisation et Services (GridUSe-2004)*, Metz, France, June 2004.

[50] E. Jeannot and F. Wagner. Message Scheduling for Data Redistribution through High Performance Network. In *École DRUIDE 2004 (DistRibUtIon de Donnée à grande Echelles)*, Le Croisic, France, May 2004.

[51] Jeannot E. Compression adatative et dynamique de donnés. In *Ecole thématique GRID 2002*, pages 55 – 63, Aussois, France, December 2002. INRIA.

[52] M. Cosnard and E. Jeannot. Automatic parallelization of coarse grained programs. In *Journées de l'informatique Messine (JIM'99)*, Ile de saulcy, Metz, France, May 1999.

[53] M. Cosnard and E. Jeannot. Ordonnancement de graphes de tâches paramétrés. In *Conception et mise en œuvre d'applications parallèles irrégulières de grande taille (ICaRE'97)*, Aussois, France, December 1997.

[54] M. Cosnard, E. Jeannot, and M. Loi. PlusPyr un outil d'aide à la parallélisation. In G. Bernard, J. Chassin de Kergommeaux, B. Folliot, and C. Roucairol, editors, *Placement dynamique et répartition de charge : application aux systèmes répartis et parallèles*, Collection didactique INRIA, dec. 1996, pages 131 – 150, Presqu'île de Giens, France, July 1996.

Rapport de recherche non publié par ailleurs

[55] Martin Do, Jack Dongarra, Emmanuel Jeannot, and Phillip J Mucci. A Test Suite for PVM. Technical Report ut-cs-95-277, Department of Computer Science, University of Tennessee, Knoxville, 1995.

Autres rapports de recherche

[56] Emmanuel Jeannot and Flavien Vernier. A Practical Approach of Diffusion Load Balancing Algorithms. Research Report 5875, INRIA, March 2006.

[57] Eddy Caron, Bruno DelFabbro, Frédéric Desprez, Emmanuel Jeannot, and Jean-Marc Nicod. Managing Data Persistence in Network Enabled Servers. Research Report RR-5725, INRIA, October 2005.

[58] E. Jeannot. Improving Middleware Performance with AdOC : an Adaptive Online Compression Library for Data Transfer. Research Report RR-5500, INRIA, February 2005.

[59] V. Berten, J. Goossens, and E. Jeannot. On the Distribution of Sequential Jobs in Random Brokering For Heterogeneous Computational Grids. Research Report RR-5499, INRIA, February 2005.

[60] E. Jeannot and F. Wagner. Modelizing, Predicting and Optimizing Redistribution between Clusters on Low Latency Networks. Research Report 5361, INRIA, LORIA, November 2004.

[61] Y. Caniou and E. Jeannot. Improvements and Study of the Accuracy of the Tasks Duration Predictor, New Heuristics. Technical Report RR-5206, INRIA, May 2004.

[62] Y. Caniou and E. Jeannot. Study of the behaviour of heuristics relying on the Historical Trace Manager in a (multi)client-agent-server system. Technical Report RR-5168, INRIA, April 2004.

[63] E. Jeannot and F. Wagner. Message Scheduling for Data Redistribution through High Performance Networks. Research Report 5077, INRIA, LORIA, January 2004.

[64] Johanne Cohen, Emmanuel Jeannot, and Nicolas Padoy. Parallel Data Redistribution Over a Backbone. Technical Report RR-4725, INRIA, February 2003.

[65] Y. Caniou and E. Jeannot. Schedulig on the GRID : Historical Trace and Dynamic Heuristics. Technical Report RR-4620, INRIA, November 2002.

[66] E. Jeannot. Adaptive online data compression. Technical Report RR-4400, INRIA, France, March 2002.

[67] F. Desprez and E. Jeannot. Adding Data Persistence and Redistribution to NetSolve. Technical Report RR2001-39, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, France, 2001.

[68] B. Cirou and E. Jeannot. Tripet : a Clustering Scheduling Algorithm for Heterogeneous Systems. Technical Report RT-0248, INRIA, France, 2001.

[69] E. Jeannot. Automatic code generation in the task graph model. Technical Report RR-1230-00, LaBRI, Université de Bordeaux I, France, 2000.

[70] M. Cosnard, E. Jeannot, and T. Yang. Symbolic Partitionning and Scheduling of Parameterized Task Graphs. Technical Report RR1998-41, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, France, September 1998. (www.ens-lyon.fr/LIP/publis.us.html).

[71] M. Cosnard, E. Jeannot, and L. Rougeot. Low Memory Cost Dynamic Scheduling of Large Coarse Grain Task Graphs. Technical Report RR98-14, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, France, March 1998.

[72] M. Cosnard and E. Jeannot. Building and Scheduling Coarse Grain Task Graphs. Technical Report RR97-03, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, France, Ecole Normale Supérieure de Lyon, France, February 1997.

Annexe B

Publications

B.1 Ordonnancement des calculs

L'article *Triplet : a Clustering Scheduling Algorithm for Heterogeneous Systems*, paru dans International Workshop on Metacomputing Systems and Applications (MSA'2001), présente notre contribution à l'ordonnancement de graphes de tâches pour les environnements hétérogènes en appliquant les techniques de *clustering*.

L'article *Multi-Criteria Scheduling Heuristics for GridRPC Systems*, publié dans la revue International Journal of High Performance Computing Applications, présente nos résultats majeurs dans le domaine de l'ordonnancement multicritère pour les systèmes GridRPC. Dans cet article nous étudions à échelle réelle différentes heuristiques en les testant dans l'environnement NetSolve.

L'article *On the Distribution of Sequential Jobs in Random Brokering for Heterogeneous Computational Grids*, publié dans la revue IEEE Transactions on Parallel and Distributed Systems présente notre approche du courtage de ressources stochastiques où l'indéterminisme et la dynamicité de l'environnement est défini par des variables aléatoires.

B.2 Transfert des données

L'article *Message Scheduling for Parallel Data Redistribution between Clusters* publié dans la revue IEEE Transactions on Parallel and Distributed Systems présente nos travaux sur l'ordonnancement de message pour la redistribution de données. Nous proposons un algorithme à performance garantie et nous étudions son comportement sur des cas réels.

L'article *Improving Middleware Performance with AdOC : an Adaptive Online Compression Library for Data Transfer* paru dans IEEE IPDPS, présente la dernière version de notre bibliothèque de compression adaptative AdOC ainsi que son évaluation dans les cas défavorables (réseau rapide, donnée déjà compressée, etc.).

L'article *Managing Data Persistence in Network Enabled Servers* paru dans la revue Scientific Programming Journal, présente les mécanismes de persistance et de redistribution de données dans les environnements de type GridRPC. Nous montrnos comment ces mécanismes on été implanté dans NetSolve et dans DIET.

B.3 Environnements pour l'expérience

L'article *Wrekavoc : a Tool for Emulating Heterogeneity*, paru dans Heterogeneous Computing Workshop (HCW 06), présente Wrekavoc, notre outil qui permet de définir et de contrôler l'hétérogénéité d'un cluster. Cet outil est évalué à l'aide de benchmark et nous montrons que la dégradation des caractéristiques physique se fait de manière indépendante.

L'article *Grid'5000 : A Large Scale And Highly Reconfigurable Experimental Grid Testbed* publié dans la revue International Journal of High Performance Computing Applications présente l'environnement GRID'5000 et en particulier l'outil OAR de réservation et l'outil de déploiement Kadeploy.

Triplet : a Clustering Scheduling Algorithm for Heterogeneous Systems

Bertrand Cirou
LaBRI, Université Bordeaux I
351, cours de la Libération
33405 Talence Cedex, France
cirou@labri.fr

Emmanuel Jeannot
LORIA, Université Nancy I
615, rue du Jardin Botanique
54602 Villers les Nancy, France
ejeannot@loria.fr

Abstract

The goal of the OURAGAN project is to provide access of meta-computing resources to Scilab users. We present here an approach that consists, given a Scilab script, in scheduling and executing this script on an heterogeneous cluster of machines. One of the most effective scheduling technique is called clustering which consists in grouping tasks on virtual processors (clusters) and then mapping clusters onto real processors. In this paper, we study and apply the clustering technique for heterogeneous systems. We present a clustering algorithm called triplet, study its performance and compare it to the HEFT algorithm. We show that triplet has good characteristics and outperforms HEFT in most of the cases.

1 Introduction

Scilab is an heavily used tool in the mathematical community [7]. As Matlab, Scilab allows to execute scripts for engineering and scientific computation. However, Scilab has some limitation since it is not parallelized. The goals of Scilab$_{//}$[1], developed in the OURAGAN project[1]is to permit an efficient and transparent execution of Scilab on a meta-computing environment. Various approaches have been taken in order to achieve these objectives. The approach we propose is the following. Given a Scilab script, first, we analyze and compute its dependencies. Second, we build a task graph that model the inner parallelism of the script. This script is then scheduled on an heterogeneous cluster of workstations. In the last step we execute this script on the cluster. In this paper we focus on the scheduling and executing steps. In the literature a lot of work has been done for scheduling task graphs to an homogeneous set of processors [2, 4]. Algorithms for homogeneous processors are inefficient for most of network of workstations (NOWs). Indeed, most of the time, NOWs are made of heterogeneous computers. Several algorithms have been proposed to tackle the problem of scheduling tasks on an heterogeneous architecture [9, 10, 13]. All these algorithms implement the list-based scheduling technique. Two-step scheduling techniques have been shown to be very efficient for homogeneous systems [6, 11, 14, 15]. A two-step scheduling algorithm works as follows. The first step is the clustering phase : tasks are grouped into clusters. The main idea of this phase is to group tasks on virtual processors in order to suppress unnecessary communications. The second phase is called the mapping phase: each cluster is assigned a processor. This technique has been very successful because the clustering phase is global. This is opposed to list scheduling algorithms where only local optimizations are performed.

The research topic concerning clustering of static task graphs in the case of an heterogeneous platform is relatively unexplored. M. Eshaghian and C. Wu have proposed an algorithm called cluster-M in [5]. However, in our opinion, cluster-M has the following deficiencies. As the progression of the clustering is done by following the topological order of the task graph, bad clusters are then built. Moreover, this clustering always embeds the most communicant task onto its father, which is not always the best way to generate parallelism. Thus, the clustering computed by the cluster-M algorithm may not always be effective. In this paper, we propose a theoretical metric which describes the behavior of a good clustering algorithm. Our main contribution is that we propose a multi-step scheduling algorithm for heterogeneous NOWs. In order to apply the clustering technique to heterogeneous systems we show that we need to cluster both tasks and machines. We show that our algorithm, called *triplet* behaves as requested by the metric. Finally, we have compared our algorithm to the HEFT algorithm [9]. It appears that, in general, for heterogeneous network of workstations, triplet outperforms HEFT.

This paper is organized as follows. In Section 2, we describe the model of task graphs and heterogeneous systems we target. In Section 3, we present the new metric. Our

[1] `http://www.ens-lyon.fr/~desprez/OURAGAN`

multi-step algorithm is presented in Section 4. The metric conformity is shown in Section 5. The complexity of our algorithm is computed in Section 6. Experimental results are described in Section 7. In Section 8 we give concluding remarks.

2 Definitions and Models

Task Graphs. We use the task graph model to model our programs. A task graph is an annotated directed acyclic graph defined by the tuple $G = (V, E, T, C)$. V is the node set, representing a task. E is the edge set. There is an edge between task i and task j if there is a dependence between task i and task j. T is the number of instructions task set. C is the communication volume set. Transforming a Scilab script into a task graph is out of the scope of this paper. For more details the reader should refer to automatic parallelization techniques [3] or to the MATCH project [8].

Heterogeneous System Model. Heterogeneous systems we target are networks of workstations as one can find in a laboratory. Each workstation can communicate to any other workstation but communication links may have different speed. Each workstation may be different and can executes tasks at different speed. Hence we model an heterogeneous system by the following tuple : $H = (S, L)$ where S is the set of machine speed. The number of machines is $|S|$. L is the link bandwidth set. There is a communication link between every machine.

Execution Model. We assume that tasks are atomic: a machine executes a single task at a time. The total amount of CPU time required to execute a task is calculated by dividing the number of instructions of the task by the power of the CPU in MIPS. For instance, executing task i on machine m requires total amount of time of $T_{\text{comp}} = t_i/s_m$ where $t_i \in T$ and $s_m \in S$. A task can start its computations only when it has received all its data and can send data only when its computations are finished. The time taken to transfer data from task i to task j between machine m and n is $T_{\text{com}} = c_{i,j}/l_{m,n}$ where $c_{i,j} \in C$ and $l_{m,n} \in L$.

3 Metric

A metric allows to class algorithms depending on solutions they produce. A well known metric on homogeneous platform is the speedup, but it looses some sense when applied to the heterogeneous case. Hence we need to find new ones for evaluating algorithms in the general case. Yarmolenko et al. proposed in [16], new criteria called efficiency and utilization. This metric only apply to independent tasks and without communication. Here are the definitions for evaluating processors: let TET be the total execution time of all tasks for a given schedule and T_{seq} the total execution time of all tasks on the best processor. Efficiency and utilization for processors are defined as follow:

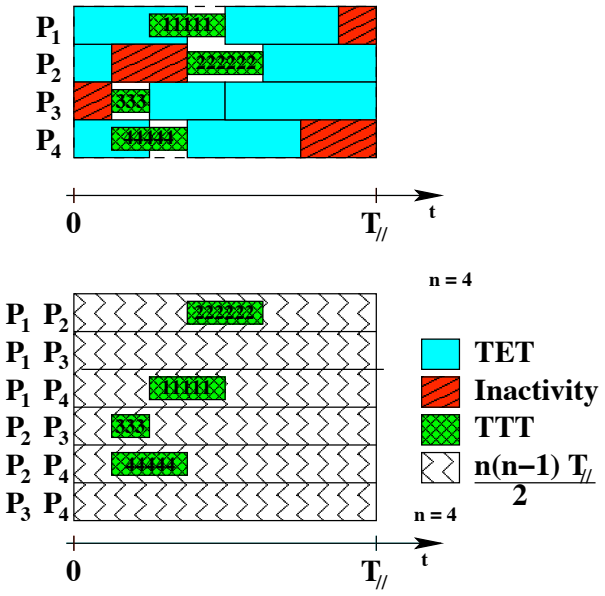


Figure 1. Gantt Chart for Processors and Network Links

$$E = \frac{T_{\text{seq}}}{TET}, U = \frac{TET}{nT_{//}}$$

Thus, $T_{//}$ can be extracted from these two equations:

$$T_{//} = \frac{T_{\text{seq}}}{nEU}$$

Since this metric does not take the network into account, we make an extension with two new formulas for evaluating communications.

In Figure 1 we present two Gantt charts, the upper one is for the tasks scheduling and the second below corresponds to the scheduling of the communications.

Let n the number of workstations, TTT be the total transfer time, TST the total of the n smallest communications time and $\frac{n(n-1)}{2}$ the number of links in the fully connected graph of processors.

TST is a constant for a given DAG and a given platform, this value is calculated by dividing the $n-1$ smallest communication by the bandwidth of the fastest link. TST is the minimal communication time spent when all the $n-1$ processors are used (the first task starts its computation on one processor and launches computations on the $n-1$ other processors with these $n-1$ communications). We set the

network efficiency and utilization for n workstations:

$$E_{\text{net}} = \frac{TST}{TTT}\ , U_{\text{net}} = \frac{2TTT}{n(n-1)T_{//}}$$

The network efficiency indicate whether only necessary communications are performed. The network utilization give an estimation of the average load of the network. If communications are present all along the execution of the program then we reach the case where the network utilization is maximal.

We can express $T_{//}$ as a function of the efficiency and the utilization. We get a new network dependent formula.

$$T_{//} = \frac{2TST}{E_{\text{net}}U_{\text{net}}n(n-1)}$$

Minimizing $T_{//}$ can be done by maximizing the product of the efficiency by the utilization. An important fact is that utilization and efficiency are divergent, the more utilization grows, the smaller efficiency is.

4 The Triplet Algorithm

Homogeneous versus Heterogeneous. Clustering algorithms have been very successful for homogeneous platforms [12, 15]. These algorithms are fast : for the best clustering algorithms the complexity is bounded by sorting edges of the task graph. Moreover, list-scheduling algorithms traverse the graph using a topological sort and therefore does only local optimizations. On the other hand, clustering algorithms consider global criterions to map tasks to clusters and then are able to perform global optimizations. Hence, it appears that in most of the cases, clustering algorithms are faster and give better results than list-scheduling algorithms. This remark motivates us for trying to build a clustering algorithm for heterogeneous platforms.

However, adapting clustering algorithms to the heterogeneous case is a difficult task. In homogeneous systems the duration of a communication depends only on the number of data exchanged and the duration of a task depends only on the number of operations to perform. In an heterogeneous system this is no longer true. Indeed, the duration of a communication depends also on the speed of the network link taken and the duration of a task depends on the processor that will execute this task. Therefore, techniques used for clustering tasks on homogeneous systems such as comparing the size of data exchanged between tasks or the number of operations of a task give little informations for an homogeneous system (large data can be exchanged rapidly on a fast link and small data can be exchanged slowly on a slow link). Hence, clustering algorithms for homogeneous systems cannot give good results on heterogeneous systems.

Since one cannot know if a given task or a given communication is going to be longer that an other task or communication prior to mapping clusters to processors, we propose to group machines that share the same characteristics (network, processor speed,etc...) and then to map clusters of task to clusters of machines. The main advantage of this approach is that these clusters of machines are sets of somehow homogeneous hardware. Hence, during the clustering phase, we can take decisions knowing that clusters of tasks are going to be mapped on relatively homogeneous clusters of machines.

Our multi-step scheduling algorithm for heterogeneous platforms is a bit different than multi-step scheduling algorithm for homogeneous platforms since it is performed in three steps.The first step is the clustering of tasks. Tasks are grouped into clusters in order to suppress unnecessary communications while preserving parallelism. The second step is the workstation clustering. As mentioned above, in order to efficiently map clusters to machines, machines which are somehow equivalent need to be grouped together. In the last step, task clusters are mapped to workstation clusters.

Algorithm 1 Tasks clustering

Require: A task graph and a system topology graph
Ensure: The clustering of the task graph
1: Put each task in a cluster.
2: Generate the list of all the triplets.
3: Sort the triplets by decreasing degree and by decreasing amount of communication.
4: **for** each triplet **do**
5: **if** geometric or temporal criterion is fulfilled **then**
6: merge the two clusters
7: **end if**
8: **end for**

Clustering Tasks. Algorithm 1 is our task clustering algorithm. Initially, tasks are put in different clusters. In order to suppress unnecessary communications we need to merge some clusters. For merging clusters our algorithm considers tasks which belong to a path of length 2 in the task graph. Every path of length 2 is composed of three tasks and is called a *triplet*. We consider triplets of tasks instead of pairs of tasks because there are many more triplets than pairs. Thus, our algorithm test merging possibilities more often along the growth of clusters.

Before starting the clustering phase, triplets are all generated. Then, they are all considered one at a time. Therefore, we need to sort triplets in order to consider large communicating edges first. Triplets are sorted first by their degree and second by their decreasing amount of communication produced by its three tasks. Hence, our algorithm will first

clusterize parts of a task graph that presents few parallelism and high communications costs.

Our algorithm considers each triplet. Let t_2 and t_4 be two tasks of a triplet with t_2 a predecessor of t_4 and respectively belonging to cluster C_1 and C_2 (see Figure 2). Cluster C_1 and C_2 are merged if one of the two following criteria is true.

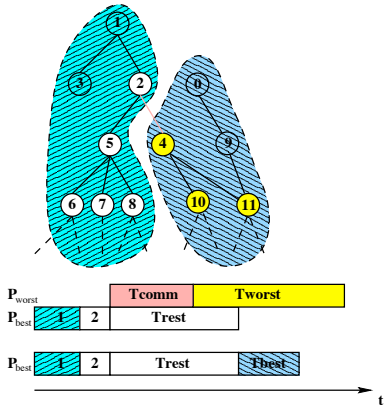


Figure 2. Temporal criterion

Figure 2 shows the first criterion that is based on temporal parameters. Let T_{rest} be the time needed to compute all successors of t_2 that are in C_1 on the best processor. Let T_{worst} be the time required to execute on the worst processor all successors of t_4 that are in C_2. Let T_{best} be the time to execute all tasks of C_2 on the best processor. Let T_{comm} be the duration of the communication between t_2 and t_4 evaluated on the worst network link. Our first criterion cause the merging when $T_{\text{comm}} + T_{\text{worst}} < T_{\text{rest}} + T_{\text{best}}$. This means C_1 and C_2 are not merged only if it worth not merging in the worst case. This criterion controls the width of cluster by suppressing parallelism each time there is a risk to be slower if the two clusters are on different processors. The second criterion use geometric properties of the clusters. Two clusters C_1 and C_2 are merged if they do not overlap more than 20% and the resulting cluster is at least 20% greater than initials clusters. This criterion is purely morphological, its main goal is to keep clusters elongated.

Clustering Workstations. After having done task clustering, we clusterize the workstation graph as well to get a global view of the problem. This clustering is needed for detecting machines that present the same characteristics. We suppose we deal with LAN type network, where each computer is able to do point to point communications. The clustering is done by sorting machines, then by going through the sorted list and creating a new cluster each time the variation between two consecutive workstations is big enough.

Algorithm 2 Mapping task's clusters onto clusters of workstations

Require: A set of task's clusters and a set of workstation's clusters

Ensure: Assign a Workstation to each task

1: Sort workstation's clusters by decreasing network capabilities and by CPU power.
2: Determine a maximum load for each cluster of workstation.
3: Sort task's clusters by amount of external communications and by number of instructions.
4: **for** each task's cluster in the order **do**
5: **if** if the number of operations assigned to current cluster of workstations exceed its load. **then**
6: switch to next workstation cluster.
7: **end if**
8: Assign current task's cluster to the Workstation having the best completion time.
9: **end for**

We sort workstations, first by decreasing network capabilities, then by decreasing CPU power. Clusters are defined in the following way: while two consecutive workstations have less than 10% of difference concerning their bandwidth and CPU power we add them on the same cluster. In the other case we create a new cluster with the last workstation considered. This clustering is fast and permits to put together computers that are somewhat equivalent.

Mapping Tasks Clusters onto Workstations Clusters. Our mapping algorithm is shown in Algorithm 2. Each task cluster has its own values for the amount of output communication and the total number of instructions required to achieve. Clusters of workstations are labeled with their overall network capabilities and total CPU power. Before doing the mapping, we need to sort clusters of each sort (first by the network parameter and second by the computation parameter). Moreover, the mapping must equitably load each cluster of workstations, hence we need the representation of each of these clusters compared with the others. Our mapping algorithm allocates task's cluster, in the order, to workstations having the best completion time as long as the load limit is not exceeded. This mapping insure that the largest communications are done on the best links and each cluster of processors has nearly the same time computation load.

5 Metric Conformity

The triplet algorithm contributes to minimize $T_{//}$. In the two formulas we got for $T_{//}$, we need to maximize the product of the efficiency by the utilization. E is improved at the

assignment time, because we choose the processor that minimize execution finish time of the cluster being mapped. U is maximized thanks to the load balancing done between clusters of workstations.

Concerning the network, the reduction of $T_{//}$ depends of the product $E_{net}U_{net}$. During the clustering, only communications that are profitable are kept. Thus, E_{net} is high. For U_{net}, the geometric criterion increase the number of tasks executable at time t. Hence, the probability of communications is high and then the value of U_{net} too.

6 Complexity

Let be Δ_i^+ and Δ_i^- the in and out degree of the task i and Δ_{max}^+, Δ_{max}^- the maximum in and out degree of the graph. Let be N_t the number of triplets in the DAG. Each task i contributes to $\Delta_i^+\Delta_i^-$ triplets : for one incoming edge there is one triplet per outgoing edge. Thus N_t is the sum of all these triplets for each task in the DAG.

$$N_t = \sum_{i=1}^{|V|} \Delta_i^+\Delta_i^- < \sum_{i=1}^{|V|} \Delta_{max}^+\Delta_{max}^- = |V|\Delta_{max}^+\Delta_{max}^-$$

In the worst case, $\Delta_{max}^+ = \Delta_{max}^- = O(|V|)$ and the number of triplets is $O(|V|^3)$. However for most of the graphs Δ_{max}^+ and Δ_{max}^- are constant and small. For some graphs only the maximum out degree *or* the maximum in degree is related to $|V|$ (In the Gaussian Elimination task graph, for instance $\Delta_{max}^+ = 2$ and $\Delta_{max}^- = 0(\sqrt{|V|})$. Let $\Delta_{max} = \Delta_{max}^+\Delta_{max}^-$. The complexity of the clustering phase is bounded by a sort of all the triplets, hence its complexity is $O(|V|\Delta_{max}\log(|V|\Delta_{max}))$.

Let p be the number of processors. For the mapping step, the worst case is reached when we have only one cluster of tasks and one of workstations. For each task, the best processor is taken among the p ones available. The mapping takes $O(p|V|)$. The triplet algorithm takes $O(|V|\Delta_{max}\log(|V|\Delta_{max}) + p|V|)$.

Since for most of the task graphs Δ_{max} is a constant, we claim that our algorithm has a very competitive complexity.

7 Results

We have implemented a task graph execution simulator for testing various heterogeneous topology. We use another task scheduling algorithm: HEFT [9] (Heterogeneous Earliest Finish Time) to make a performance comparison with our triplet algorithm. The HEFT algorithm is based on evaluating the shortest path of execution to a terminal task.

We define the processor heterogeneity and network heterogeneity as follows:

$$H_{\text{proc}} = \frac{\text{standard deviation of CPU power}}{\text{average of CPU power}}$$

$$H_{\text{net}} = \frac{\text{standard deviation of network bandwidth}}{\text{average of network bandwidth}}$$

If we take 10 PCs at 333Mhz, 5 PCs at 800Mhz and 5 PCs at 1GHz, then $H_{\text{proc}} \sim \frac{292.18}{616.5} \sim 43.4\%$ If we replace the 10 PCs at 333Mhz by others at 166MHz, the heterogeneity reaches 70%

These definitions allow to compare the heterogeneity of recent workstation network with older ones, because we have the average value in the formula that has a normalization effect. For our benchmarks we have generated $40\ 000$ task graphs, and $40\ 000$ different topologies with varying heterogeneity. We have fixed the heterogeneity of the bandwidth at 50%, on the other hand the processor heterogeneity takes the following values: 31%, 36%, 42%, 48%. Values indicated in Figure 3 correspond to the middle of the interval inside which each parameter is randomly chosen. We expose here only a part of our results, we have in fact 16 histograms with various task graph heterogeneity. As they are quite similar we chose not to present them. The y-axis of the results shown Figure 3 is the time our solution is faster than HEFT solution : we outperform HEFT each time the bargraph is greater than 1. At the first look we see that the performance of our triplet algorithm increases with the heterogeneity. Our algorithm manages the heterogeneity better than HEFT does, thanks to our multi-step approach: clustering then mapping. We have evaluated our metric by checking wether or not there could be schedules that have poor efficiency and utilization but produce good makespan. We conducted 3000 tests for HEFT and triplet on various DAGs of different heterogeneity and got the result that the algorithm that have the best makespan always have the best product of efficiency by utilization.

8 Conclusion and Future Work

We have presented a new algorithm for scheduling task graph on a heterogeneous system of workstations. Our contribution is three-fold. First, The algorithm is based on a multi-step approach that allows global optimizations. Second, we have presented a new metric that express requirements a good scheduling algorithm must have and we have shown that our algorithm fulfills these requirements. Lastly, we have compared our algorithm to the well-known and efficient HEFT algorithm. In most of the cases, our algorithm outperforms HEFT.

We want our execution scheme to adapt to load unbalance that appears when other users run applications on the cluster. In order to do that we plan to migrate tasks within its machine clusters – a machine cluster is composed of similar machines – during execution.

Finally, we plan to incorporate this algorithm to Scilab$_{//}$ in order to execute Scilab script on clusters of workstations.

References

[1] E. Caron, S. Chaumette, S. Contassot-Vivier, F. Desprez, E. Fleury, C. Gomez, M. Goursat, E. Jeannot, D. Lazure, F. Lombard, J. Nicod, L. Philippe, M. Quinson, P. Ramet, J. Roman, F. Rubi, S. Steer, F. Suter, and G. Utard. Scilab to Scilab$_{//}$, the OURAGAN Project. *To appear in Parallel Computing*, 2001.

[2] P. Chretienne and C.Picouleau. *Scheduling Theory and its Applications*, chapter 4, Scheduling with Communication Delays: A Survey, pages 65–89. John Wiley and Sons Ltd, 1995.

[3] M. Cosnard and M. Loi. Automatic Task Graph Generation Techniques. *Parallel Processing Letters*, 5(4):527–538, 1995.

[4] H. El-Rewini, T. Lewis, and H. Ali. *Task Scheduling in Parallel and Distributed Systems*. Prentice Hall, 1994.

[5] M. M. Eshagian and Y. C. Wu. Mapping heterogeneous task graphs onto heterogeneous system graphs. *Heterogeneous Computing Workshop (HCW'97)*, pages 147–160, April 1997.

[6] A. Gerasoulis and T. Yang. On the Granularity and Clustering of Direct Acyclic Task Graphs. *IEEE Transactions on Parallel and Distributed Systems*, 4(6):686–701, June 1993.

[7] C. Gomez, editor. *"Engineering and Scientific Computing with Scilab"*. Birkhäuser, 1999.

[8] M. Haldar, A. Nayak, A. Kanhere, P. Joisha, N. Shenoy, A. Choudhary, and P. Banerje. Match Virtual Machine: An Adaptive Runtime System to Execute MATLAB in Parallel. In *International Conference on Parallel Processing (ICPP-2000)*, Toronto, Canada, Aug. 2000.

[9] S. H. Haluk Topcuoglu and M.-Y. Wu. Task scheduling algorithms for heterogeneous processors. *8th IEEE Heterogeneous Computing Workshop (HCW'99)*, pages 3–14, April 1999.

[10] M. Kafil and I. Ahmad. Optimal task assignment in heterogeneous computing systems. *Heterogeneous Computing Workshop*, pages 135–146, April 1997.

[11] Y.-K. Kwok and I. Ahmad. Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):506–521, May 1996.

[12] J.-C. Liou and M. A. Palis. A New Heuristic for Scheduling Parallel Programs on Multiprocessor. In *IEEE Intl. Conf. on Parallel Architectures and Compilation Techniques (PACT'98)*, pages 358–365, Paris, Oct. 1998.

[13] A. Radulescu and A. van Gemund. Fast and effective task scheduling in heterogeneous systems. In *Proceeding of Heterogeneous Computing Workshop*, 2000.

[14] V. Sarkar. *Partitionning and Scheduling Parallel Program for Execution on Multiprocessors*. MIT Press, Cambridge MA, 1989.

[15] T. Yang and A. Gerasoulis. DSC Scheduling Parallel Tasks on an Unbounded Number of Processors. *IEEETPDS*, 5(9):951–967, Sept. 1994.

[16] V. Yarmolenko, J. Duato, D. K. Panda, and P. Sadayappan. Characterization and Enhancement of Static Mapping Heuristics for Heterogeneous Systems. Technical Report OSU-CISRC-02/00-TR07, Dept. of computer science, Ohio State University, Feb. 2000. To be presented in HiPC'2000.

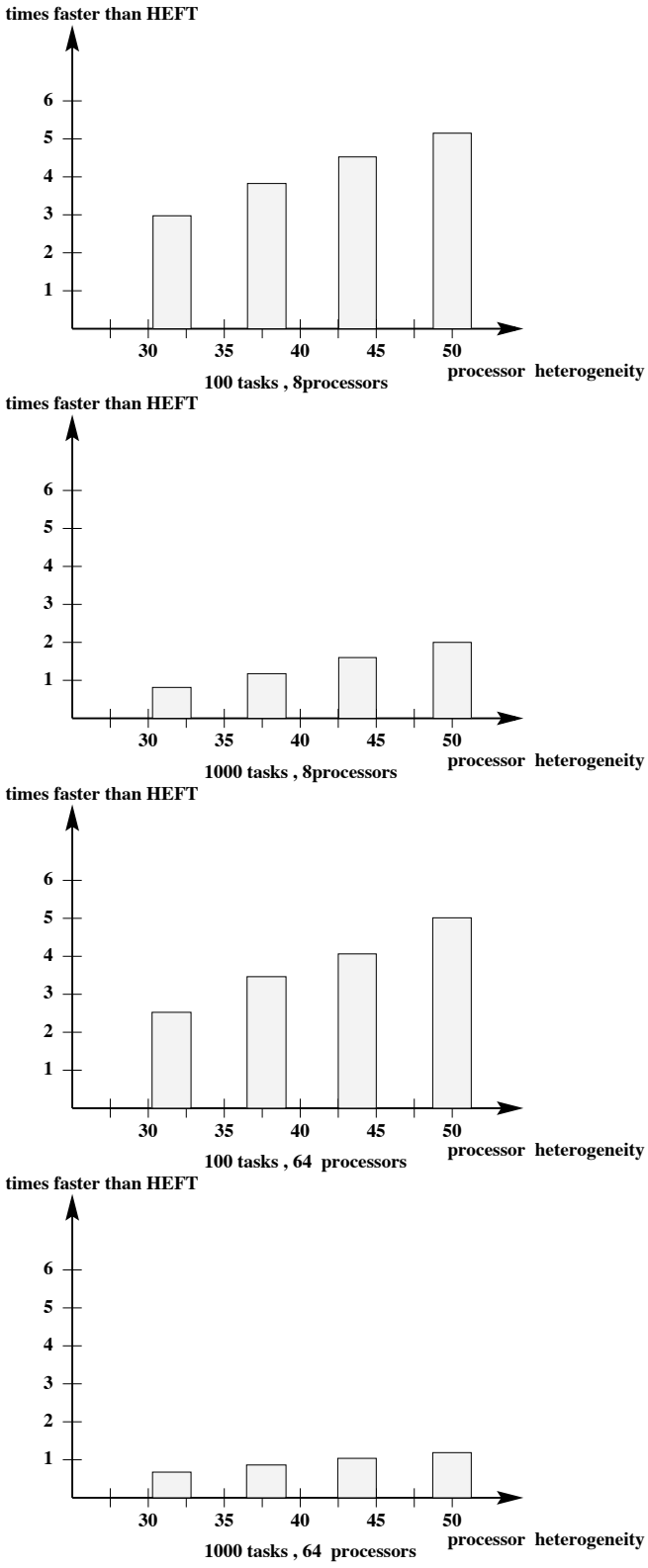


Figure 3. Average values: Communication 50Ko, Task 500 000 instructions, Bandwidth 10 Mbits/s, CPU 300MHz

MULTICRITERIA SCHEDULING HEURISTICS FOR GRIDRPC SYSTEMS

Yves Caniou[1]
Emmanuel Jeannot[2]

Abstract

In this paper, we address the problem of dynamically scheduling independent tasks and/or application task graphs on a GridRPC environment. Resources are assumed to compute submitted jobs within the time-share model. We present a non-intrusive predictive module, the historical trace manager (HTM), which is able to give the completion date of each task in the system. Four heuristics relying on its estimations are proposed and compared to the well-known minimum completion time (MCT) algorithm. We first analyze the accuracy of the HTM. Then we show with an extensive simulation study, and with numerous scenarios of execution performed on a real-world platform, that our heuristics outperform MCT on several metrics among which are the makespan and the response time.

Key words: Dynamic scheduling heuristics, multicriteria scheduling, grid computing, experimental evaluation

1 Introduction

GridRPC (Nakada et al. 2003) is an emerging standard promoted by the Global Grid Forum (GGF; http://www.ggf.org). This standard defines both an API and an architecture. A GridRPC architecture is heterogeneous and composed of three parts: a set of clients, a set of servers, and an agent (also called a registry). The agent is charged with mapping client requests to servers. To ensure that a GridRPC system is efficient, the mapping function must choose a server that will fulfill several criteria. First, the choice must allow the total execution time of the client application to be as short as possible. Secondly, each request of every client must be served as fast as possible. Finally, the resource utilization must be optimized.

Several middlewares instantiate the GridRPC Model: NetSolve (Casanova and Dongarra 1996), Ninf (Nakada, Sato, and Sekiguchi 1999), DIET (Caron et al. 2002), etc. In these systems, a server executes all its assigned requests when received and never delays the start of the execution. In this case, we say that the execution is time-shared (in contrast to space-shared when a server executes at most one task at a given moment). In NetSolve, the scheduling module uses the minimum completion time (MCT; Maheswaran et al. 1999) heuristic to schedule requests on the servers. MCT was designed for scheduling applications on space-shared servers. The goal was to minimize the makespan of a set of independent tasks. Thus, MCT chooses the server where the last request finishes the soonest. Indeed, within the user-space model, the completion date of the last task is less than or equal to the overall makespan. Minimizing each task's completion date implies the minimization of the overall set of tasks. However, due to interferences of the execution of two tasks, this is no longer true in a time-share environment. Furthermore, this scheduling strategy leads to the following drawbacks.

- *Mono-criteria and mono-client*. MCT was designed to minimize the makespan of an application. It is not able to give a schedule that optimizes other criteria such as the response time of each request. Furthermore, optimizing the last task completion date does not necessarily minimize the makespan of individual clients. Indeed, in the context of GridRPC, the agent has to schedule requests from more than one client and it has no information about the application to which the task belongs.

[1]UNIVERSITÉ CLAUDE BERNARD LYON 1

[2]INRIA-LORRAINE LORIA (UMR 7503, CNRS, INPL, INRIA, UNIVERSITÉ HENRI POINCARÉ NANCY 1, UNIVERSITÉ NANCY 2

The International Journal of High Performance Computing Applications, Volume 20, No. 1, Spring 2006, pp. 61–76
DOI: 10.1177/1094342006061890

- *Load balancing*. MCT tries to minimize the execution time of each request. This leads to overuse of the fastest servers. In a time-share environment, this implies delaying previously mapped tasks and therefore degrades the response time of the corresponding requests.

MCT requires sensors that give information on the system state. It is mandatory to know the network and server states in order to make good scheduling decisions. However, actively monitoring the environment is intrusive and perturbs its performance. Moreover, information is passed to the agent only periodically. These data may be invalid by the time the scheduling decision is made.

In order to address these drawbacks, we propose and study four scheduling heuristics designed for GridRPC systems. Our approach is based on a prediction module that runs only on the agent. This module is called the historical trace manager (HTM), and it records all scheduling decisions. Because it runs on the agent, it is not intrusive and there is no delay between the determination of the state of the system and the availability of the information. The HTM takes into account the fact that servers run under the time-share model. It is able to predict the duration of a given task on a given server as well as its impact on already mapped tasks. In this paper, we show that the HTM is accurate and is able to predict very precise task durations on moderately loaded servers. The four proposed heuristics use the HTM to schedule the tasks.

Our approach is an experimental one. We have performed both simulation experiments and real-platform tests. In order to perform the tests on a real platform, we have introduced the HTM and our heuristics into the NetSolve system and performed intensive series of tests on a real distributed platform (almost two months of continuous computation) for various experiments with several clients.

In Maheswaran et al. (1999), MCT was proposed and studied only in the context of independent task submission (called a metatask). In this paper our study is more general: tests contain submissions of independent tasks as well as submissions of task graphs (one-dimensional meshes and stencil). Moreover, we consider several kinds of application graphs in the parallel job scenario and independent tasks are submitted during the execution of the graphs. We do not assume any knowledge of the graphs and we schedule dynamically each request. The goal is to best serve each client request (be it a parallel task graph or a single independent task).

We have compared our heuristics against MCT implemented in NetSolve on several criteria (makespan, sumflow, meanflow, etc.). Results show that the proposed heuristics outperform MCT on a majority of these criteria with simultaneous performance gains, up to 20% for the makespan and to 60% for the average response time.

2 MCT in GridRPC Systems

The GridRPC model (also called the client–agent–server model) is an extension of the client–server model where the agent dispatches client requests on already registered servers.

In this model, the agent is the critical component. It knows the state of the environment and schedules client requests on servers that are able to execute them. Servers are computational distributed resources. Each server, once launched, contacts the agent and gives its list of problems it is able to solve. Finally, a client is a program that requests computational resources. It asks the agent to find a set of the most suitable servers that are able to solve its problems.

In this approach, the agent is launched first, then the servers can register with the agent by sending the list of problems they are willing to solve as well as their peak performances and network capabilities. The client, who has a computational need, contacts the agent, which, in return, gives him a ranked list of servers. The client sends the request to each server in turn using an RPC-like call until one agrees to resolve the corresponding task. The client then sends the input data and the execution of the task begins as soon as the transfer is completed. When the request has finished, the server sends back the output data to the client.

In this framework, the performance of the whole system depends greatly on the scheduling heuristic implemented in the agent and the accuracy of the information the agent has on the system.

NetSolve (Casanova and Dongarra 1996) instantiates the GridRPC model. A NetSolve agent uses MCT (Maheswaran et al. 1999) to map tasks on servers. The MCT heuristic chooses the server that will finish the task the fastest. In order to determine the completion time of a given task on a server, it assumes that the load on the server will be constant during the execution of the task. This leads to the following remarks.

1. The agent needs an evaluation of each server load.
2. The load on a given server is seldom constant. If the server is loaded by some tasks, they are likely to finish during the computation of the new mapped task.
3. The time-share model implies that mapping a task on a loaded server delays the currently executing tasks. The perturbation caused by the new task on the others may make previous decisions obsolete.

First, the load evaluation of a network or of a machine is a difficult task and is often performed by sensors. NetSolve can use NWS sensors (Wolski, Spring, and Hayes 1999) or its own. It faces two major problems: accuracy and intrusiveness.

Sensors run short processes on servers in order to evaluate the current load. It needs some CPU cycles and therefore delays other processes. In such an architecture, the delay is important only if the probe rate is high. However, relatively frequent measurements are needed in order to obtain a reasonable accuracy.

The accuracy problem is the major problem. Indeed, a sensor sends to the agent, at given intervals, the load of the server it observes. Between these intervals, the load may change, and thus cause the information on the agent to become inaccurate.

Secondly, MCT considers that the load is constant during the execution of a task. Therefore, it can make misguided scheduling decisions. For instance, if two identical servers are equally loaded, MCT has no other information to determine on which one to schedule a new task. However, if for some reason it is known that on one of these two servers a task will finish very soon, a good heuristic would a priori choose this server.

Thirdly, mapping a task on a server delays the other running tasks. This delay is called the "perturbation" in this paper. It is very important to take the perturbation into consideration. Indeed, delaying tasks can make former decisions obsolete ("server *X* has been chosen because it leads to shorter execution time than server *Y*"). Conversely, we need to ensure that, if we map a task to a server, there will not be too many tasks that will delay this one.

3 Historical Trace Manager

The HTM is an attempt to efficiently answer the three remarks exposed in the previous section. It is a prediction module that runs on the agent. It is accurate and non-intrusive. It simulates the execution of the tasks on each server and therefore is able to predict load variation and to compute the perturbation of existing tasks caused by the introduction of new tasks.

3.1 Time-Share Model

At a given moment it is possible that a server has to run more than one job. This happens, for instance, when the system is heavily loaded or when the set of servers is heterogeneous (in this case, for performance reasons, the agent may often select the fastest servers). This is true even if servers are dedicated to the grid middleware.

We use the following model to simulate time-shared resources. When n tasks are using the same resource (CPU, network, etc.), each one uses $1/n$ of its peak power.

3.2 HTM Algorithm

The GridRPC standard imposes that each request is divided into three parts: first, the transmission of the input data, then the computation on the server, and finally the transmission of the output data. The HTM performs a discrete event simulation for these three parts. In order to do this, it needs several types of information: server and network peak performances, the size of input and output data and the number of operations of each task. All this information is static and can be computed off-line (note that MCT also needs this information). Therefore, the HTM answers the three remarks exposed in Section 2.

1. The HTM is not intrusive because it uses only static information. Furthermore, because it runs on the agent, information is immediately available for scheduling heuristics. The accuracy of the information given by the HTM is very high and will be experimentally demonstrated in Section 6.
2. The HTM is able to compute the completion date of each task in the system, in particular of a new request, by simulating its execution. Therefore, the load is not assumed to be constant. For instance, if two identical servers are equally loaded, it knows the remaining duration on each request on each server and therefore is able to optimize the choice of the server to map a new request.
3. The HTM can simulate the mapping of a task on any server. Hence, it can compute the perturbation of this task on all currently running tasks. The perturbation is different for each server and can therefore be used for making scheduling decisions, as we will see in the next section.

3.3 Notations

We use the following notations. a_i is the arrival date of task i. T_i' is the simulated finishing date in the current system state and C_i is the real one (post-mortem). The HTM can simulate the execution of a new task n and give the new simulated completion dates T_i of all tasks i, $i \leq n$. We define for all $k \leq n$, $\delta_k = T_k - T_k'$, the perturbation the task n produces on each running ones (Figure 1). We also define for all $k \leq n$, $D_k = T_k - a_n$, the remaining duration of the task k before completion. $p(i)$ is the server where the task i is mapped and d_i is its duration on the unloaded server. Note that these notations are related to a given server j. However, since in the following the server number is always known, we deliberately ignore subscript j.

3.4 Usefulness of the HTM

Here follows an example that shows how the HTM can help in making good scheduling decisions.

Let us suppose that the set of servers is made up of two identical servers (the same network capabilities, the same CPU speed peak, the same set of problems, etc.). At time

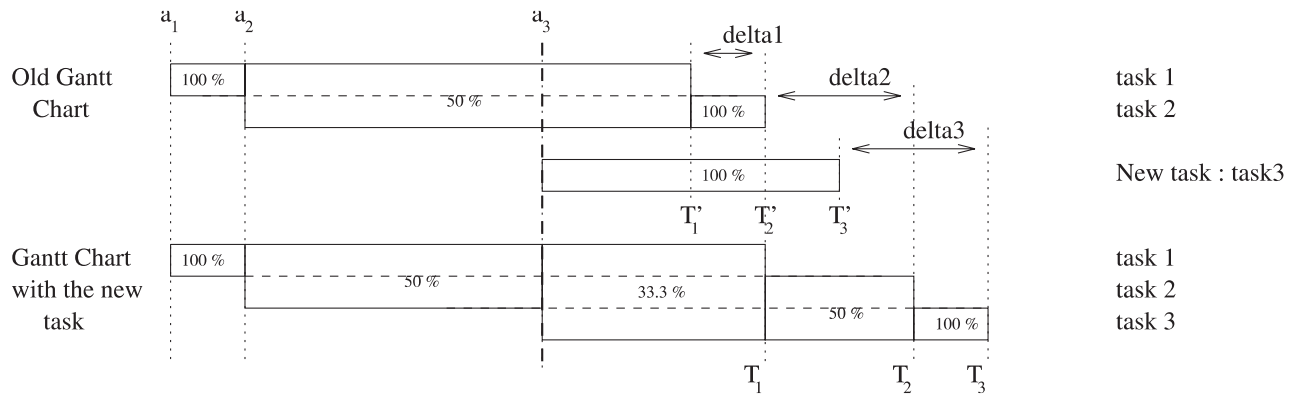


Fig. 1 Notations for the HTM. Top: task 3 is submitted to the server at time a3 while it is running two tasks. Bottom: once scheduled on the server this leads to perturbations δ_i. Percentages show the amount of CPU available for each task.

0, the client sends to these servers two tasks 1 and 2, whose durations on each server are 100 and 1000 s, respectively, with no input data. Let the agent schedule task 1 on the server 1 and task 2 on the server 2. At time 80, let a client request that the agent schedule a task 3 whose duration is 100 s.

Without the HTM, the agent knows only that server 1 and server 2 have the same load, and therefore is not able to decide which is the best server to schedule task 3 (in practice, as there is dynamic information and as the evolution of the load average is not necessarily exactly the same on the two machines, the decision is blurred).

However, the HTM simulates the execution of the tasks on each server and the agent knows that, at time 80, the remaining duration of task T_1 is 20 s while the remaining duration of task T_2 is 920 s. Therefore, it knows that scheduling T_3 on server 1 will lead to a shorter completion time than scheduling T_3 on server 2.

4 Heuristics

We introduce here four new heuristics: historical MCT (HMCT), minimum perturbation (MP), minimum sum flow (MSF), and minimum length (ML). These are compared to MCT in the following sections. The HTM simulates the new task on each server, which is able to solve the problem and gives resultant information to the scheduler. Therefore, the heuristic considers the perturbation the new task will induce on each running one, and computes the "best" server accordingly.

1 **For each** new task t
2 **For each** server j
3 Ask the HTM to simulate t on server j
4 Affect to $T_{t,j}$ the termination date of task t
5 Map task t to server j_0 such as $T_{t,j_0} = \min_j T_{t,j}$
6 Tell the HTM that task t is allocated to server j_0

Fig. 2 HMCT Algorithm.

4.1 Historical Minimum Completion Time

HMCT is the MCT algorithm relying on the HTM. For each new arriving task, the HTM simulates the mapping and the execution of the task on each server. Therefore, we have an estimation of the finishing date of this task on each server. The agent then maps the task to the server that minimizes its finishing date (Figure 2).

4.2 Minimum Perturbation

Scheduled by MP, the new task is mapped to the server j that minimizes the sum of all the perturbations that the new task induces on the environment. In the case of equality, for instance when the first task is submitted to the agent, the server that minimizes the completion date of the last incoming task is chosen (Figure 3).

```
1 For each new task t
2   For each server j
3     Ask the HTM to simulate t on server j
4     Compute P_j = Σ_i δ_i
5   If all P_j are equal
6     Map task to server j_0 that minimizes T_{t,j}
7   Else Map t to server j_0 such as P_{j_0} = min_j P_j
8   Tell the HTM that task t is allocated to server j_0
```

Fig. 3 MP Algorithm.

```
1 For each new task t
2   For each server j
3     Ask the HTM to simulate task t on server j
4     Compute P_j = Σ_i δ_i + T_t − a_t
5   Map t to server j_0 such as P_{j_0} = min_j P_j
6   Tell the HTM that t is allocated to server j_0
```

Fig. 4 MSF Algorithm.

4.3 Minimum Sum Flow

The heuristic uses the HTM to compute the sum of all the flows when assigning the last task t to each server. Hence, the heuristic returns the identity of server j_0 that minimizes the system sumflow, e.g. $\min_j(\Sigma_i(T_i - a_i))$, if assigning the task to that server. However, as the difference between two values is only due to perturbations and to the new simulated task duration, the heuristic only needs to compute $\Sigma_i\delta_i + T_t - a_t$ for each server j, that is to say, the perturbation of the last task on the server plus the manager estimated length of the new task (Figure 4). This heuristic is equivalent to minimize total interference (MTI) proposed by Weissman (1996).

4.4 Minimum Length

For each new arriving task, ML requests that the HTM simulate the execution of the request on each server. After each simulation, ML computes the quantity $\Sigma_i D_i$.

```
1 For each new task t
2   For each server j
3     Ask the HTM to simulate task t on server j
4     Compute P_j = Σ_i D_i
5   Map t to server j_0 such as P_{j_0} = min_j P_j
6   Tell the HTM that t is allocated to server j_0
```

Fig. 5 ML Algorithm.

Then, it chooses the server that minimizes all these values, e.g. the server on which the sum of the remaining duration of each task, including the new one (which is its actual flow), is the minimum (Figure 5).

5 Criteria

In this section, we present metrics that have been observed when comparing our heuristics against a modelization of MCT. In GridRPC middlewares, it is important to improve the completion of the application but also the resource utilization and the quality of service for each request. This is why we do not only observe the makespan metric. Thus, we study a number of metrics from a number of fields including system environments and continuous stream. Here are the criteria we observe for each heuristic.

- The makespan. This is the completion time of the last finished task, $\max_i C_i$. The makespan is much more an application metric, for it is its completion date. So, although it is widely utilized, basically with the MCT heuristic in Legion (Grimshaw and Wulf 1997) and NetSolve (Casanova and Dongarra 1996), we do not think it is the appropriate metric to use when considering GridRPC. The agent serves more than one user, so the agent does not necessarily deal with a single application, and must do its best for each of them.
- The sumflow (Baker 1974). This is the amount of time that the completion of all tasks has taken on all the resources, $\Sigma_i(C_i - a_i)$. Executing tasks on servers has a cost proportional to the time it takes. We can therefore consider it as a system and economics metric, for it estimates the utilization of the resources and the profit realized by using a given heuristic when the cost of each resource is the same.
- The meanflow, also called the response time. In our context, it is the average duration of a task in the system, given by the value $\Sigma_i(C_i - a_i)/n$ if n is the number of tasks submitted in the system. This metric is mostly considered for independent tasks and focuses on the quality of service that the system is able to deliver to each request.
- The maxflow (Bender, Chakrabarti, and Muthukrishnan 1998). This is the maximum time a task has spent in the system, $\max_i(C_i - a_i)$. In a loaded system, a task will generally execute longer than expected. This is even more true if it is allocated on a fast server (which is generally more solicited). This value can account for high resource contention and promote better load balance in heterogeneous platforms;
- The maxstretch (Bender, Chakrabarti, and Muthukrishnan 1998). This metric denotes the maximum factor, $\max_i((C_i - a_i)/d_i)$, by which a query has been slowed down relative to the time it takes on the same but

Table 1
Two time-share model experiments.

Task	Arrival date	Size of the matrix	Real completion date	Simulated completion date	Difference	Percentage of error
1	33.00	1500	80.79	79.99	0.8	1.7
2	59.92	1200	92.08	93.19	–1.11	3.4
3	73.92	1800	142.79	142.50	0.29	0.4
1	29.41	1500	76.69	76.29	0.4	0.8
2	56.43	1200	89.15	89.50	–0.35	1
4	96.41	1200	136.97	139.40	–2.43	5.9
6	140.41	1200	204.84	204.85	–0.01	0.02
3	70.42	1800	210.61	195.74	14.87	10.6
5	121.43	1500	235.38	232.92	2.46	2.2
8	181.45	1200	248.02	248.56	–0.54	0.8
9	206.41	1200	259.91	261.63	–1.72	3.2
7	166.42	1800	289.08	288.91	0.17	0.1

unloaded server. A client can have an approximation of the minimum time its task will take on a server, but a task can require much more time than it would due to contention with previously allocated tasks and with hypothetical arriving tasks. This value gives the worst case of slowdown for a task among all those submitted to the agent.

- The number of tasks that finish sooner. Whereas this is not a metric, this value gives, in correlation to the previous metrics, a relevant idea of a quality of service given to each task when comparing two heuristics. For instance, comparing the heuristics H_1 with MCT (on the same set of tasks $\{t_1 \ldots t_n\}$ and the same environment), it is $\left|\{t_i | (C_{t_{iH_1}} < C_{t_i MCT})\}\right|$.

A typical user is generally not interested in minimizing the completion time of the final task in the system (optimizing the makespan) but rather that his own tasks (a subset of all client requests) finish as fast as possible. Therefore, if we can provide a heuristic where most of the tasks finish sooner than if scheduled by MCT without delaying too much other task completion dates (which can be verified with the meanflow for example), we can claim that this heuristic, from the user point of view, outperforms MCT.

6 Validity of the Time-Share Model and Accuracy of the HTM

In this section, we compare the durations of real tasks against their estimation made by simulation in the HTM.

We tackle two objectives. First, we determine the validity of the time-share model and therefore the limits of simulation. Secondly, we evaluate the accuracy of the HTM.

First, we have experimented using the time-share model on Linux and Solaris systems when tasks are matrix multiplications and have the same priorities (two examples, where tasks are ranked by their completion date, are given in Table 1). The percentage error is defined by 100 multiplied by the absolute value of the difference divided by the real duration of the task. We have seen small variations between the simulated and real completion dates (a mean of less than 3% with regard to the duration). We can see that in this test, at most five tasks are running at the same time on the server (tasks 3, 5, 6, 7, and 8 between time 181.45 and time 204.84). Task 3 is the most impacted task as all the tasks run during its execution (it started while task 1 had not yet finished and ended after the start of task 9).

Next, we performed tests to measure the accuracy of the model as the number of concurrent processes changes. In order to perform these tests, we have integrated our HTM into the code of the agent in NetSolve. We have performed 100 experiments where 500 non-identical jobs have been requested. Indeed, a job can require 20 s, 35 s, or 50 s on the unloaded server. The real and the HTM estimated durations of each task during each experiment have been recorded. In Figures 6, 7 and 8, we observe the following.

- In dark dots: the ratio for each task of the HTM estimated completion date divided by the real post-mor-

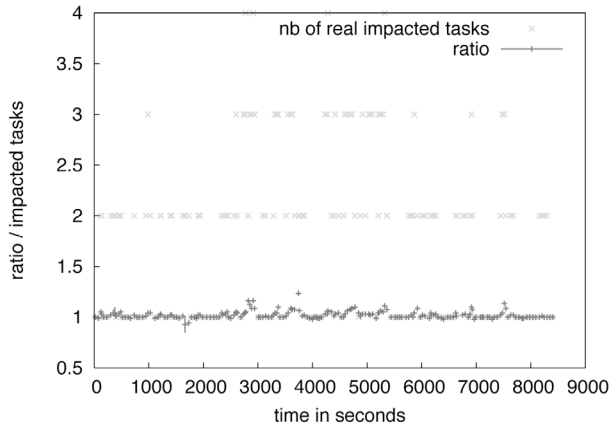


Fig. 6 Independent tasks submission with MP.

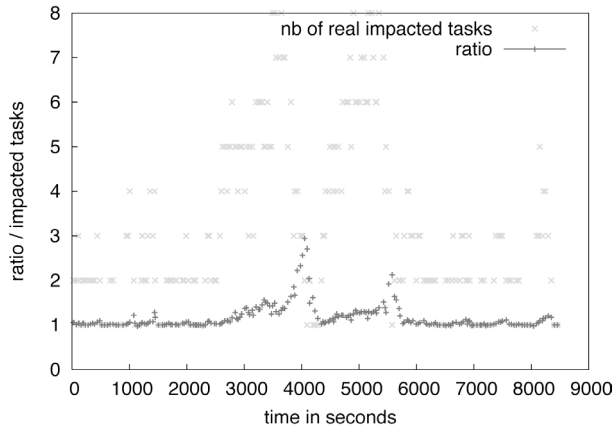


Fig. 7 Independent tasks submission with MSF.

tem one, indexed by the submission date on the *x*-axis. Hence, the closer to 1 the ratio is, the most accurate the prediction is.

- In light dots: the number of tasks that have interfered during the task execution.

We have found three different behaviors.

In Figure 6 we see that the HTM predicts accurate completion dates of previously assigned and still running tasks, taking into account interferences tasks have on each other.

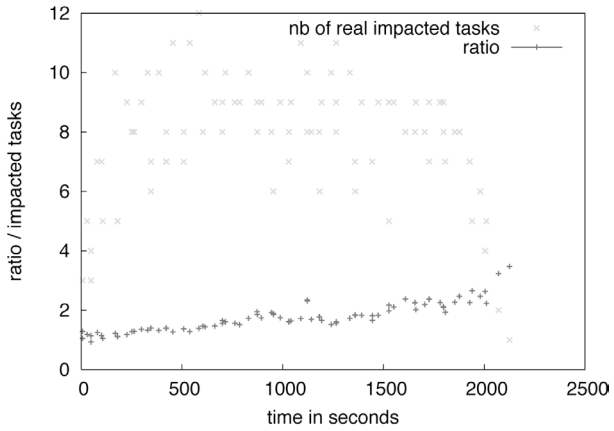


Fig. 8 Task graph submission with HMCT.

In Figure 7, estimations are degrading with the load of the server. Indeed, until 2800 s, estimations are more than 96% accurate on average and still 93% accurate until 3156 s. Then, more than six tasks are simultaneously executed on the server, thus increasing prediction errors. The HTM regains a high accuracy when the load decreases. The desynchronization and resynchronization of the HTM occur either when the rate is much too fast for the environment or when the heuristic tends to overload some servers. We should also note that the HTM always predicts tasks flow greater than in reality.

The results in Figure 8 correspond to an execution of numerous task graphs on the environment. We observe that HTM estimations are always higher than real durations. A high number of tasks, more than six tasks on average, are executed by the server at the same time. This induces high delays for the tasks scheduled on this server and creates errors. Due to precedence relationships, errors are cumulative, causing the accuracy of the predictions to degrade with time. This occurs only for some specific scheduling heuristics that overload servers with independent tasks.

Because the HTM is an environment simulator, the results induce that independent tasks simulations are relevant to foresee what can be expected in reality. Indeed, the HTM and the Simgrid tool (Casanova 2001) for example use similar simulation mechanisms. None the less, there are some limitations due to the arrival rate, to the heterogeneity (tasks and servers) and to the heuristic: accuracy is obtained if less than five tasks are executed simultaneously on a server.

In conclusion, concerning simulating tasks in the time-share model, as performed by the Simgrid tool (Casanova 2001) for instance, we see that the accuracy of the model depends on the number of simultaneously running tasks. When too many tasks are running at the same time, the time-share model overestimates the task duration. Therefore, the HTM, which is based on this model, is highly accurate when the number of simultaneous running jobs is lower than five. When this number is greater than six, the accuracy may degrade. For independent tasks submission, the HTM is able to remain highly accurate and its accuracy does not degrade later on. In the case of some task graphs and for some specific heuristics, the error is cumulative. Despite this, we see in the next section that heuristics based on the HTM are able to give good results.

7 Independent Tasks Submission: Simulation Results

In the previous section, we have shown that when submitting a set of independent tasks, the time-share model is reasonably accurate to allow simulations.

7.1 Simulated Environment

A problem solver environment is simulated with the Simgrid tool (Casanova 2001). The dynamic mapping heuristics are evaluated using some parameters that characterize heterogeneous servers and each task of the set of independent tasks. We explain in this section the different models that we used for client–agent–server mechanisms, heterogeneous entities and independent tasks. We used the Gnu Standard Library (Galassi and Theiler 1996) for all the probabilistic distributions used thereafter.

We assume the agent has perfect knowledge of the following information:

- current server load;
- current network load;
- peak CPU and network performances;
- number of operations of any tasks;
- size of the input and output data of any tasks.

MCT needs all this information while the HTM (and, consequently, our heuristics) needs only the static information (the last three items). Static information is easier to compute accurately as compared to dynamic information (for example, task durations can be obtained from benchmarks or by means of executions performed on each server; Quinson 2002). Therefore, in our simulations, the performance that MCT will obtain using this information will be better than in any real GridRPC middleware.

7.1.1 Platform Model Characterization We assume that the client and the agent are able to reach each server. Experiments are conducted on the basis that servers are dedicated to the environment (for instance, a set of clusters where reservation is possible). Moreover, we suppose that all problems can be solved on any server. We do not consider any arrival/withdrawal of any server in the system.

7.1.2 Application Model Characterization We consider independent task submissions (tasks that have no precedence relation), requested by one or more users. A task cannot be preempted: it cannot be suspended or removed to be scheduled on another server.

A task begins to be executed on a server as soon as the total amount of the input data has been received by the server. We consider that a task is finished when the data output is completely received by the client. We want to test how the heuristics react under different environments and we want to achieve the best possible overview when comparing each of them against MCT. Therefore, the same environments and the same task sets are generated for each heuristic; comparisons are conducted on the same sets of tuples (servers, tasks, arrival dates, etc.).

Experiments are conducted with the number of servers held at 25. They have one processor and their computing capacity is drawn from a uniform distribution in the range [150, 000, 500, 000]. Their network cards have the same performance (100 Mbit/s).

We use a uniform distribution to draw from the range [1 Kb, 300 Mb] the input and output sizes of data to be transferred between the client and the server. The computation cost is generated from a uniform distribution, with the following rules.

- The computation phase costs more than 10 times the transfer phase.
- The computation phase must not be greater than 600 s on the fastest server of the 25 available.

To draw task arrival dates, we use a Poisson distribution whose parameter μ varies from 0 to 70. Simulation results indicate a practical limit at $\mu < 10$; for these rates, MCT schedules more than five tasks on a server, and the results are consequently distorted. In contrast, when $\mu = 70$, each server is executing at most one task at a given moment for half of the heuristics tested. For each value of μ, we have varied the number of tasks from 10 to 250. Each plot in a graph is the mean of results of 1000 different tasks sets; each heuristic required 200,000 simulations.

7.2 Results

Results are given in Figures 9, 10, 11, and 12. Each is composed of five three-dimensional graphics, one for

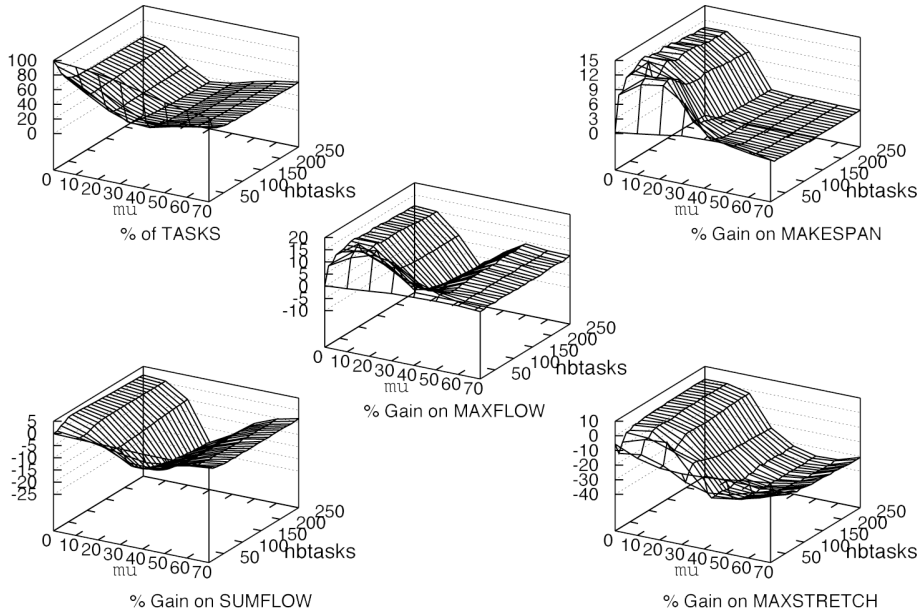


Fig. 9 Results for HMCT versus MCT on 25 servers.

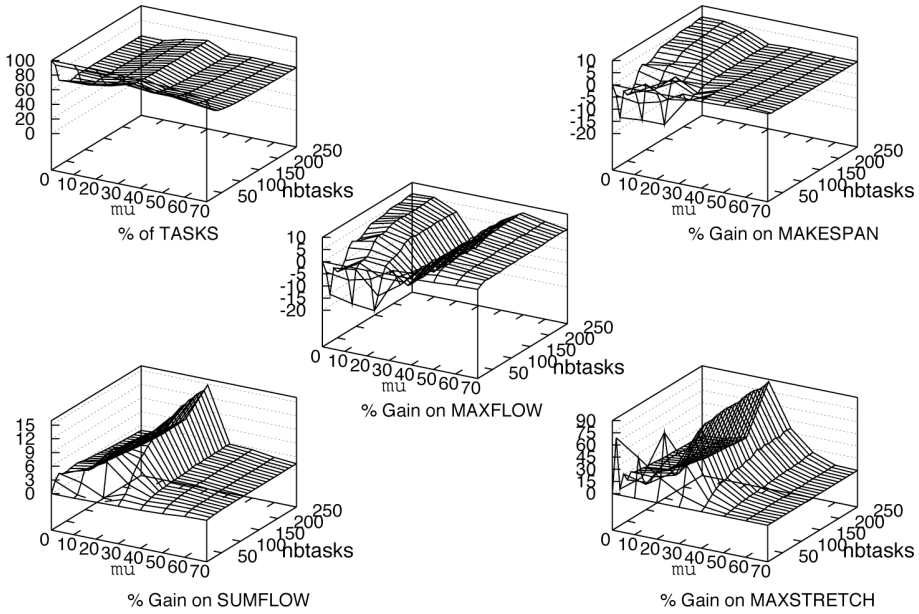


Fig. 10 Results for MP versus MCT on 25 servers.

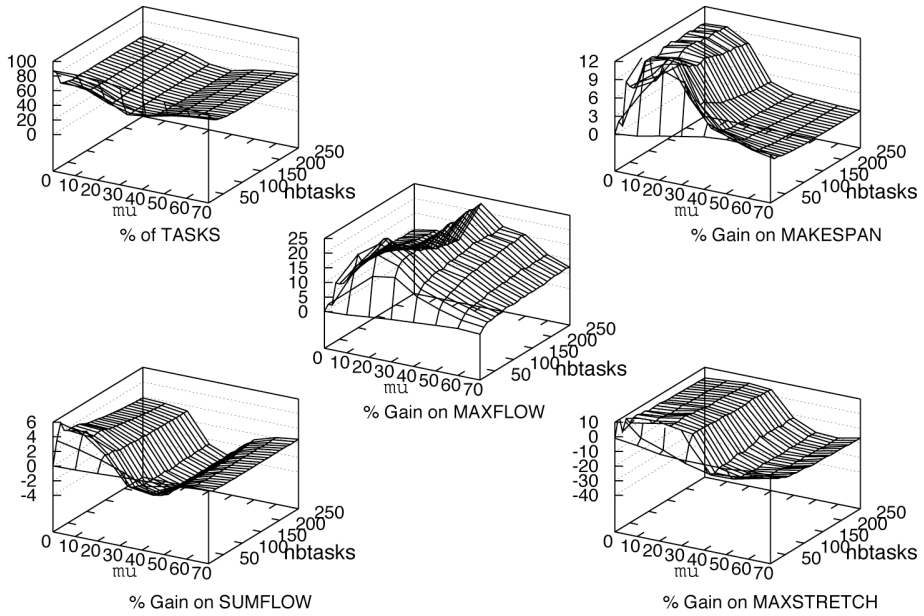


Fig. 11 Results for MSF versus MCT on 25 servers.

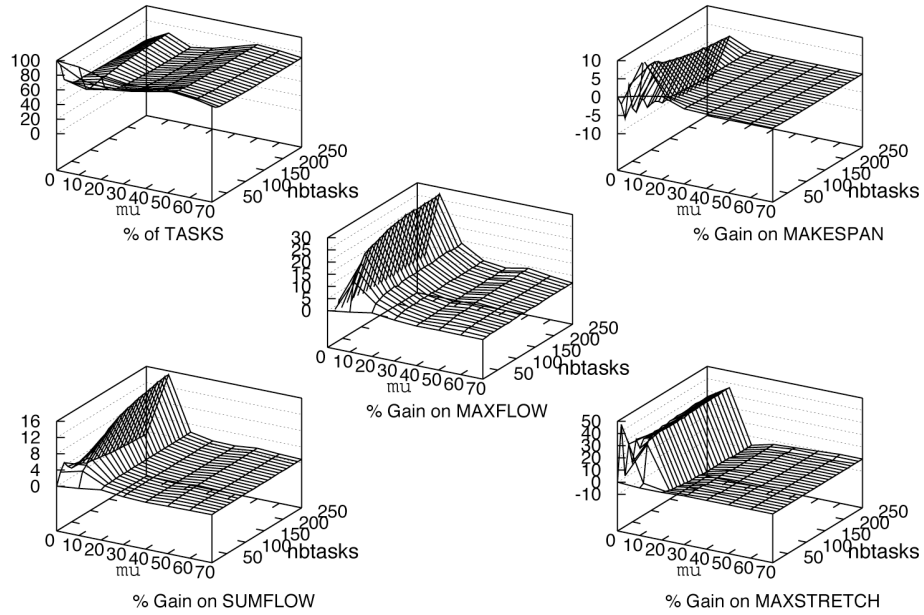


Fig. 12 Results for ML versus MCT on 25 servers.

each metric. They present the gain (in percentage) of the heuristic over MCT. For example, the first graph shows the percentage of "tasks that finish sooner". It shows a gain if the value is greater than 50. In the others, a benefit is realized as soon as the percentage is positive. Note that because of the type of submission, the gain demonstrated on the sumflow is the same as that obtained on the response time.

Considering that the makespan of an application composed of independent tasks is mainly due to the last completed task starting time (Maheswaran et al. 1999; Caniou and Jeannot 2002), we cannot expect a priori a great gain on the makespan.

To compute server scores in our simulations, MCT uses information which is far more precise than in a real environment such as NetSolve. In consequence, our modelization of MCT behaves better than the algorithm does in reality. Therefore, if we build a heuristic that outperforms our simulation of MCT, this heuristic will certainly outperform MCT in reality.

7.2.1 Historical Minimum Completion Time The goal of this heuristic is the same as that of MCT: it expects to minimize the makespan of the application by minimizing the completion date of incoming tasks.

Figure 9 shows that there is a gain greater than 8% on the makespan for $\mu \leq 20$. For $\mu \geq 30$, the gain is still positive even if almost null. The sumflow is greater for HMCT than MCT for $\mu > 10$ (leading to degraded performance). Because HMCT tends to optimize the use of fast servers for new tasks, it delays the running ones (for $\mu \leq 10$, the rate is so high that the flow is increased for both MCT and HMCT). It also results in a percentage of "tasks that finish sooner" lower than 50% (under 20% for $\mu = 30$). For $\mu \geq 40$, the maxstretch shows that a task is 30% longer than if scheduled with MCT; there is contention for fast servers, even at a low rate.

The main drawback of this heuristic is that it tends to overload the fastest servers, which has two effects: it unnecessarily delays task completion dates and, in a highly heterogeneous environment, servers may collapse, mainly due to a lack of memory.

7.2.2 Minimum Perturbation MP aims to provide a better quality of service to each task by delaying as little possible already allocated tasks.

Figure 10 shows that when $nbtask > 80$, the gain on the makespan is positive (greater than 5% for $nbtask = 250$ and $\mu < 30$). MP allows gain on the sumflow, with a peak at 15% for $\mu = 30$. The percentage of "tasks that finish sooner" is always greater than 60%, with a peak at 70% for $\mu = 30$ (when, in most cases, no more than one task is running on a given server). As a result of the use of slower servers, there is always a gain on the maxstretch; in some cases, the gain reaches 90%. The maxflow is always better except for $\mu = 30$.

Despite these good results, MP presents a major drawback: when only one server is idle, it is chosen regardless of its speed, possibly jeopardizing its performance. This occurs when dealing with highly heterogeneous resources and/or a high rate of costly requests; for example, when $\mu < 30$ and $nbtasks < 50$, MP is outperformed by MCT on the makespan.

7.2.3 Minimum Sum Flow Minimum sum flow is an attempt to mix the advantages of HMCT and MP (to keep the makespan objective of HMCT, with less risk of collapsing fastest servers, and to give a better quality of service to each task) and to reduce the cost of resources.

MSF performs well on the makespan, around 9% better when $\mu \leq 20$, against MCT (Figure 11). The percentage of "tasks that finish sooner" is slightly worse than that of MCT for $25 \leq \mu \leq 50$, but is always above 40%. For $\mu \geq 35$, tasks are slowed down by a maximum factor of 20% (occurring when $50 \leq \mu \leq 60$). Sumflow performances depend on the parameter μ: MSF maximizes the use of fast servers when $30 \leq \mu \leq 40$ and thus delays tasks. It induces a greater flow cost. However, when $\mu \leq 30$, MCT overloads faster servers and MSF, taking into account interferences, performs slightly better than MCT.

7.2.4 Minimum Length Except for $\mu = 0$, the gain on the makespan is always positive (Figure 12). ML has also a positive gain on the sumflow, with a peak at 15% when $\mu = 10$. On the maxstretch, MCT performs slightly better for $20 \leq \mu \leq 40$, yielding around 60% on the percentage of "tasks that finish sooner", with a peak at 80% for $\mu = 50$.

ML achieves a makespan at least as good as MCT for $\mu \neq 0$, an adequate performance on the maxstretch and gains on the sumflow, the maxflow and on the percentage of "tasks that finish sooner". As the mapping decision is partly made using the cost of the new task, ML does not exhibit MP's drawback. Therefore, this heuristic tends to overcome the drawbacks of HMCT and MP, while keeping the advantages of both.

7.3 Discussion

To summarize our heuristics performances, we can first conclude that our heuristics outperform MCT on the makespan. However, the percentage of "tasks that finish sooner" of HMCT is poor and the maxstretch is always negative. In a client–agent–server context, this needs to be improved. MP shows great performances on every metrics but can present the drawback that it unnecessarily utilizes slow servers. MSF behaves better than MCT. Indeed, it combines HMCT and MP performances. It outperforms ML on the makespan and on the maxflow, but loses on the other

Table 2
Experimental platform.

Type	Machine	Processor	Speed	Memory	Swap	System
Server	spinnaker	xeon	2 GHz	1 Go	2 Go	Linux
	artimon	pentium IV	1.7 GHz	512 Mo	1024 Mo	Linux
	soyotte	sparc Ultra-1		64 Mo	188 Mo	SunOS
	fonck	sparc Ultra-1		64 Mo	188 Mo	SunOS
Agent	xrousse	pentium II bipro	400 MHz	512 Mo	512 Mo	Linux
Client	zanzibar	pentium III	550 MHz	256 Mo	500 Mo	Linux

metrics. ML tends to overcome the drawbacks of HMCT and MP in addition to delivering good performances on every metrics, except when all tasks are submitted at the same time (e.g. $\mu = 0$, an almost impossible situation).

Because ML has positive results on all observed metrics and outperforms all the other heuristics on the percentage of "tasks that finish sooner than MCT", we consider that overall it is the best among all the tested heuristics for this section.

8 Task Graphs Submission: Real Platform Results

In Section 6, we have shown that the time-share model may not be very accurate when submitting task graphs. Therefore, we have tested our heuristics on task graphs on a real platform.

The HTM and the heuristics HMCT, MP, MSF, and ML have been implemented in the NetSolve agent in order to study the accuracy of the HTM predictions and to compare the heuristics with the MCT algorithm present as a default scheduling heuristic in NetSolve (Casanova and Dongarra 1996).

8.1 Experiments

Several experiments have been conducted in a heterogeneous environment whose resources are given in Table 2.

Three types of tasks have been used for these experiments. The duration of each task on each server is given in Table 3. Tasks are computing-intensive and require less than 1 s of data transfer.

Table 4 gives a summary of the series of tests that we have performed. Eight scenarios (from a to h) have been submitted to the platform. Two types of graphs have been used: one-dimensional meshes and stencil graphs. Each stencil graph is submitted by at most one client, while numerous one-dimensional meshes are submitted by different clients (five or 10). Stencil graphs have a width of five or 10 and a depth ranging from 25 or 50.

Table 3
Duration of the tasks on the unloaded servers.

Server	Task 1	Task 2	Task 3
spinnaker	15	30	43
artimon	17	33.5	49.5
soyotte	128	256	382.5
fonck	127.5	254	380.9

In scenarios (c), (f), (g), and (h), independent tasks are submitted by other clients during the execution of the graph. The duration of each task has a uniform probability (Grimshaw and Wulf 1997) to be of one of the three types given in Table 3. In this case, inter-arrival rate is drawn from a Poisson distribution of parameter μ.

Each scenario is randomized by a different seed and runs six times per heuristic. Six runs were sufficient for the average values given in Section 8.2 to be stabilized. This leads to 720 experiments and more than 55 days of computation.

8.2 Comparison Between Heuristics

For each scenario we compare each heuristic on three criteria. MCT is used as the comparison baseline.

First, we compare the application completion time, e.g. the makespan; this comparison is done for all the scenarios.

Secondly, when independent tasks are involved (scenarios c, f, g, and h), we compare the average response time, e.g. in our context (see Section 2) the mean flow of the task. The response time is only measured for independent tasks (the response time of a graph is its makespan).

Thirdly, when independent tasks are involved we give the percentage of these that finish sooner if scheduled

Table 4
Scenarios, modalities and number of experiments.

Scenario	Application(s)		Independent tasks		Experiment	
	nbclients	width × depth	nbtasks	μ (s)	nbseeds × nbrun	total nbtasks
(a) 1D mesh	10	1 × 50	–	–	4 × 6	500
(b) 1D mesh	10	1 × variable	–	–	4 × 6	500
(c) 1D mesh + 250 independent tasks	5	1 × 50	250	20	4 × 6	500
(d) stencil (task 1)	1	10 × 50	–	–	1 × 6	500
(e) stencil (task 3)	1	10 × 50	–	–	1 × 6	500
(f) stencil + 175 independent tasks	1	10 × 25	175	28	2 × 6	425
(g) stencil + 87 independent tasks	1	10 × 25	87	40	4 × 6	337
(h) stencil + 87 independent tasks	1	5 × 25	87	25	4 × 6	212

with our heuristic than with MCT. This, in relation to the response time, gives information about the quality of service proposed by the heuristic. Indeed, if a heuristic is able to give a good response time and if a majority of tasks scheduled with this heuristic finish sooner than if scheduled with MCT, then the client sees a better quality of service with this heuristic.

Figure 13 shows the average gain on the makespan performed by each heuristic over MCT for each client and for each scenario. We see that HMCT, MSF, and ML always outperform MCT up to 25%. MP outperforms HMCT, MSF, and ML when applications are one-dimensional mesh graphs (scenarios a, b, and c). However, MP shows negative performance for some stencil graph scenarios. This is explained as follows. As a result of its design, MP can unnecessarily map a task on a slow server, for example when it is the only idle one. This delays the completion of the graph because some critical tasks are mapped on slow servers. The client must wait for the completion of these tasks before being able to resume the execution of the application.

Figure 14 shows, for each heuristic, the average gain on the response time that each task perceives when scheduled with the considered heuristic. We see that HMCT,

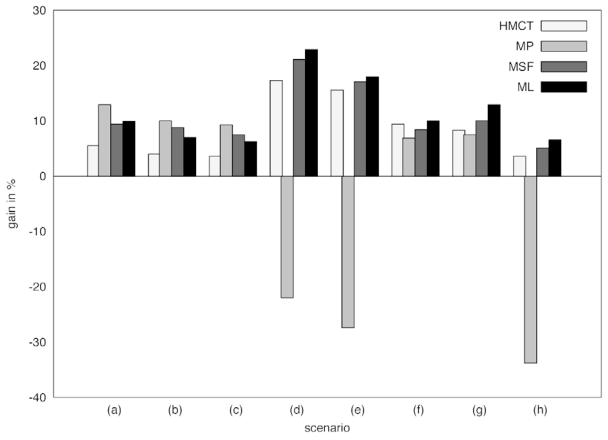


Fig. 13 Gain on the makespan for each client of each scenario.

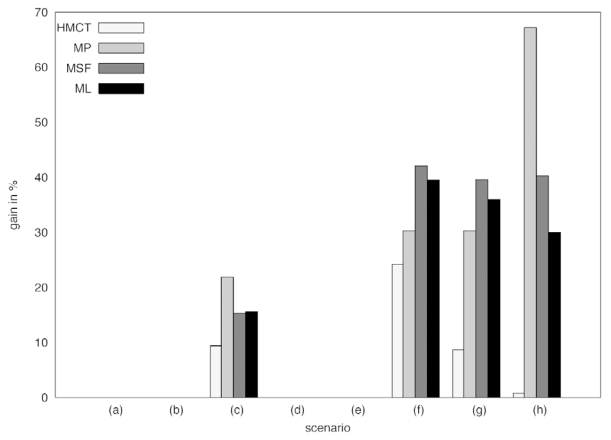


Fig. 14 Gain on the response time for each client of each scenario.

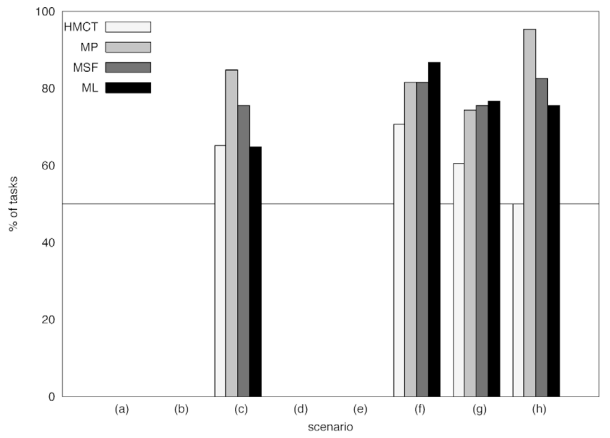


Fig. 15 Percentage of tasks that finish sooner for each scenario.

MP, MSF, and ML outperform MCT for all the scenarios. For scenarios (f) and (g), MSF is the best heuristic with a gain up to 40%. For scenarios (c) and (h), MP is the best heuristic, with more than 65% of gain in scenario (h). For this scenario, HMCT shows nearly no gain. ML achieves a poorer performance than MSF, except for scenario (c), where it gives slightly the same performance.

The average percentage of tasks that finish sooner than MCT is given for each heuristic in Figure 15. In contrast to the two former figures where a gain is observed as soon as the value is positive, the percentage here has to be superior to 50% to express a gain. We see that, as for the response time, all of our heuristics always outperform MCT. ML is the best heuristic for scenarios (f) and (g). MP is the best heuristic for scenarios (c) and (h). For scenario (h), about 90% of all the tasks finish sooner when scheduled with MP than if scheduled with MCT. For this scenario, HMCT has the same performance as MCT.

Figures 14 and 15 show together that MP, MSF, and ML are able to offer a good quality of service to each independent task. This means a task scheduled by one of these three heuristics, on average, finishes sooner and the average gain is high.

Experiment (h) shows that MP is outperformed by MSF and ML on the makespan but outperforms both of these on the quality of service. This is because MP is better at scheduling independent tasks and the two others are better at scheduling application graphs. Indeed, because of the use of slower servers by MP, the submission of the graph can suffer dramatically when waiting for a critical task to complete, thus benefiting independent tasks which profit from the whole platform. Because MSF and ML take into account for their decisions the flow that the new task will require, they use slow servers only under heavily loaded conditions.

In conclusion, in a heterogeneous GridRPC environment, MSF and ML give the best results regardless of the makespan of applications or response time of independent tasks. They achieve a higher number of tasks finishing sooner than when they are scheduled with MCT. This is more beneficial to each client, whose concern is that his own tasks, a subset of all tasks in the environment, finish as fast as possible. Depending on the main metric that the environment must optimize, ML can be employed to prefer application completion over independent tasks and MSF for the opposite, both giving good results in the other metrics.

Furthermore, we see that the "heuristic hierarchy" is conserved between simulation and real experiments. Both ML and MCT are shown to combine MP and HMCT advantages without the drawbacks.

9 Conclusion

In this paper, we have addressed the problem of scheduling tasks in GridRPC systems. Our work relies on a non-intrusive predictive module, the HTM, which simulates the execution of tasks on the environment with the time-share model.

We have presented four heuristics: HMCT, which is MCT using the far more precise HTM estimations; MP, which tries to minimize the delays that tasks induce on each other; MSF, which tries to minimize the sum of all task durations; ML, which only considers the remaining duration of all the tasks at the submission date of the new request (including it).

We have shown that the HTM is precise for independent tasks. When dealing with task graphs, the HTM is accurate when the server load is not too high. In some cases, the HTM accuracy degrades during the execution. This is due to the error accumulation. It can regain accuracy without synchronization, depending on the heuristic and the rate/kind of submission.

Our approach is an experimental one. Almost two months of continuous computation (for simulation and real world tests) have been performed in order to evaluate our heuristics.

We have tested two types of submission: independent tasks and task graphs. For independent tasks, we have performed simulation experiments. For task graphs, the time-share model is not accurate enough; we have tested our heuristics on a real platform.

Our heuristics were compared to MCT. MCT is a widely used heuristic designed for a mono-client environment where servers execute tasks sequentially, which is not the case in most GridRPC environments.

We have performed an extensive simulation study on five metrics and various parameters. Our heuristics show better performance against the idealized version of MCT, which has a far better understanding of the environment than in reality. In this case, ML is the best overall heuristic.

For our real world tests, we have used NetSolve, a GridRPC environment, in which we have implemented the HTM and all of the four heuristics. We have performed an extensive study of our heuristics on a real platform, facing the same set of experiments. Numerous scenarios involving different types of submission, with different application graphs running concurrently with an independent task workload, have been submitted to the modified NetSolve agent.

Results show that our heuristics outperform MCT on at least two of the three observed metrics: makespan, response time, and percentage of tasks finishing sooner. Depending on the main metric the environment must optimize (thus depending on the type of average submissions it will encounter), MSF or ML appear to be the best overall heuristics as they outperform MCT on all the criteria and do not present any drawbacks, as MP does.

In conclusion, ML is shown to be the best of the proposed heuristics if the application makespan is preferred to the quality of service that the environment can possibly give to each request and MSF in the opposite case. Both of these still give good performances on the other metrics.

In our future work, we will implement a mechanism to synchronize the HTM to reality. This mechanism will be based on the task completion date. The goal is to limit the effect of error accumulation seen in some experiments. Early developments show an improvement in the stability of HTM prediction accuracy.

Acknowledgment

This work is partially supported by the Region Lorraine, the French ministry of research ACI GRID.

Author Biographies

Yves Caniou received his Ph.D. in computer science in 2004 from the Universite Henry Poincare of Nancy. He received his B.S. from the Faculte des Sciences of Reims in mathematics and in computer science in 2000. He is currently Associate Professor of Computer Science at the Universite Claude Bernard, Lyon 1. His research focuses on scheduling on metacomputing platforms. He is interested in distributed scheduling algorithms, transparent batch and parallel submissions via grid environments and performance predictions.

Emmanuel Jeannot has been Associate Professor at the Universite Henry Poincare, Nancy 1, since September 2000. He obtained his Ph.D. and Master degrees in computer science (in 1996 and 1999, respectively) both from Ecole Normale Superieure de Lyon and both under the advisory of Michel Cosnard. He is performing research at the INRIA-CNRS LORIA laboratory. His main research interests are scheduling for heterogeneous environments and grids, data redistribution, grid computing softwares, adaptive online compression and programming models.

References

Baker, K. R. 1974. *Introduction to Sequencing and Scheduling*, Wiley, New York.

Bender, M. A., Chakrabarti, S., and Muthukrishnan, S. 1998. Flow and stretch metrics for scheduling continuous job streams. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, San Francisco, CA, January 25–27.

Caniou, Y. and Jeannot, E. 2002. Dynamic mapping of a metatask on the grid: historical trace, minimum perturbation and minimum length heuristics. Technical Report 4620, LORIA, Nancy, October.

Caron, A., Desprez, F., Fleury, E., Lombard, F., Nicod, J.-M., Quinson, M., and Suter, F. 2002. *Calcul réparti à grande échelle*, chapter "Une approche hiérarchique des serveurs de calculs", Hermès Science, Paris.

Casanova, H. 2001. Simgrid: a toolkit for the simulation of application scheduling. *Proceedings of the IEEE Symposium on Cluster Computing and the Grid (CCGrid'01)*, Brisbane, Australia, May 15–18, IEEE Computer Society Press, Los Alamitos, CA, pp. 430–437.

Casanova, H. and Dongarra, J. 1996. Netsolve: A network server for solving computational science problems. *Proceedings of Supercomputing*, Pittsburg, PA.

Galassi, M. and Theiler, J. 1996. The Gnu Standard Library. Available at http://www.gnu.org/software/gsl/gsl.html.

Grimshaw, A. S. and Wulf, W. A. 1997. The legion vision of a worldwide virtual computer. *Communications of the ACM* 40(1):39–45.

Maheswaran, M., Ali, S., Siegel, H. J., Hengsen, D., and Freund, R. F. 1999. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing* 59(2):107–131.

Nakada, H., Matsuoka, S., Seymour, K., Dongarra, J., Lee, C., and Casanova, H. 2003. A gridRPC model and API for end-user applications, December. Available at https://forge.gridforum.org/projects/gridrpc-wg/document/GridRPC_EndUser_16dec03/en/1.

Nakada, H., Sato, M., and Sekiguchi, S. 1999. Design and implementations of Ninf: towards a global computing infrastructure. *Future Generation Computing Systems* 15: 649–658.

Quinson, M. 2002. Dynamic performance forecasting for network-enabled servers in a metacomputing environment. *Proceedings of the International Workshop on Perform-*

ance Modeling, Evaluation and Optimization of Parallel and Distributed Systems (PMEO-PDS'02), Fort Lauderdale, FL, April 15–19.

Weissman, J. B. 1996 The interference paradigm for network job scheduling. *Proceedings of the 10th International International Parallel Processing Symposium, Heterogeneous Computing Workshop*, Honolulu, HI, April 15–19 .

Wolski, R., Spring, N. T., and Hayes, J. 1999. The network service: a distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems* 15(5–6):757–768.

On the Distribution of Sequential Jobs in Random Brokering for Heterogeneous Computational Grids

Vandy Berten, Joël Goossens, and Emmanuel Jeannot

Abstract—Scheduling stochastic workloads is a difficult task. In order to design efficient scheduling algorithms for such workloads, it is required to have a good in-depth knowledge of basic random scheduling strategies. This paper analyzes the distribution of sequential jobs and the system behavior in heterogeneous computational grid environments where the brokering is done in such a way that each computing element has a probability to be chosen proportional to its number of CPUs and (new from the previous paper) its relative speed. We provide the asymptotic behavior for several metrics (queue sizes, slowdowns, etc.) or, in some cases, an approximation of this behavior. We study these metrics for a variety of workload configurations (load, distribution, etc.). We compare our probabilistic analysis to simulations in order to validate our results. These results provide a good understanding of the system behavior for each metric proposed. This will enable us to design advanced and efficient algorithms for more complex cases.

Index Terms—Grid brokering, multilevel scheduling, random brokering, stochastic workload, heterogeneous and distributed architecture.

1 Introduction

Grid systems are the gathering of distributed and heterogeneous resources (CPU, disk, network, etc.). They are promising infrastructures for executing large scale applications and to provide computational power to everyone. In order to hide the complexity of grids from the users, executing environments (called middleware) are to be developed. A middleware aims at providing a programming API and an execution model for the applications. It relies on a set of services that enables the control of resources, the deployment of services, the execution of applications, etc. The scheduling service is responsible for the allocation of the tasks on the distributed resources. Scheduling is, therefore, a key service for enabling efficient grids.

In this paper, we focus on one kind of grid scheduler, called a resource broker or a meta-scheduler. A meta-scheduler must dispatch client requests (jobs) onto computing elements (for instance, a cluster), each of these being composed of several computing nodes (processors). Each computing element executes its own local scheduling policy for executing the subset of the jobs assigned to it. In general, the local scheduler is called a batch scheduler. A common implementation of a batch scheduler consists in storing the jobs into a queue that follows a FIFO policy. Other techniques have been also studied; see, for instance, [9] for an overview.

In order to be efficient, a resource broker needs to know the duration of the submitted job on each node of the computing resources. This assumption is not always realistic, as the duration of the job might not be known before its completion. Indeed, this duration may depend on its parameters and data. It also requires the user to have run its job at least once and to have given the obtained duration to the broker. In this work, we propose another approach where the duration of each job is not known but is given by a random variable with a fixed mean (for instance, the duration follows an exponential law). Moreover, we do not assume that the submission times of the jobs are known in advance. We suppose that the arrival time is also drawn with a random variable (for instance, a Poisson law). This probabilistic approach is driven by the dynamic nature of grids, where it is not always possible to predict both the duration and arrival time of jobs (as in a UNIX system). The proposed resource brokering algorithm is, therefore, a randomized one.

Our probabilistic approach may appear theoretical and far from actual applications, but we believe it actually is a sufficently realistic model. For example, in [8], Hui et al. present a workload characterization of the DAS-2 multicluster supercomputer which is shown to be like the one considered in this work. See also [10] for another such example. Note also that one can argue that our model is not realistic enough to model more sophisticated platforms like the European Data Grid.[1] Clearly, our long term research goal is to study more general and realisitic models with a comparable analysis (if any). This paper presents a first step in that framework by considering a quite simple but reasonable model. Starting with a simple model is the traditional approach to tackle hard problems. In fact, the works in [5], [14], [15] consider a similar model. The presented analysis provides us a strong theoretical basis and a good understanding and intuition in order to study

- *V. Berten and J. Goossens are with the Université Libre de Bruxelles, Département d'Informatique, Boulevard du Triomphe—C.P. 212, 1050 Bruxelles, Belgium. E-mail: {vandy.berten, joel.goossens}@ulb.ac.be.*
- *E. Jeannot is with Loria INRIA-Lorraine, Campus Scientifique, BP 239, 54506 Vandoeuvre les Nancy, France. E-mail: Emmanuel.Jeannot@loria.fr.*

Manuscript received 15 Jan. 2005; revised 25 May 2005; accepted 14 July 2005; published online 28 Dec. 2005.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDSSI-0028-0105.

1. http://public.eu-egee.org/.

more general and realistic models in further works and reach our long term research goal.

Resource brokering is an important subject since many middlewares, such as EDG/EGEE, use a meta-scheduler for allocating jobs. Enabling Grids for E-sciencE (EGEE) aims at building a grid for scientists. So far, it provides a set of resources (computer or clusters: several thousand CPUs) distributed across 27 countries for executing scientific jobs. EGGE is one of the few production grids functioning seven days per week, 24 hours per day.

In this paper, we study a randomized algorithm (random brokering) for allocating stochastic workload to a grid environment similar to the EDG/EGEE one. This algorithm works as follows: The probability for allocating a job to a computing element is proportional to its relative speed and its number of CPU. This strategy is not just a purely random one, and it is driven by the intuition that, the more powerful a computing element is, the more jobs it can execute. To the best of our knowledge, this strategy has never been studied in the context of grid computing. Furthermore, it is mandatory to understand such a strategy before being able to develop more complex ones. As in EGEE, we assume that the computing elements of the target grid are heterogeneous. However, inside each computing element, the processors are supposed to be homogeneous. Two cases are analyzed. In the first case, we assume that the grid is not saturated (it submitted less jobs than it can execute). In the second case, the grid is saturated (it submitted more jobs than it can execute). Each job is assumed to be sequential (requires only one CPU).

For each case, we compute the average queue length of each computing element (Section 3). The queue length is an important metric since it helps to design and tune the local batch scheduler. We also study the utilization of the grid: the number of CPUs used on each computing element (Section 4). When, due to change of the load, the grid goes from the saturated case to the nonsaturated case, we compute an approximation of the time required to empty the queues on each computing element (Section 5). Finally, jobs may be delayed depending on the load of the grid. We study this last metric (called the slowdown) for several distributions of the tasks duration (Section 6). Each of these metrics is studied both analytically and with simulation experiments. Therefore, the contribution of this paper is an extensive comprehension of the behavior of random brokering on heterogeneous computational grids.

This paper generalizes the results presented in [1], which consider *homogeneous* grids (where CPUs have all the *same* speed). In the present paper, we generalize those results by analyzing *heterogeneous* grids (where CPUs have *different* CPU speeds). We extend in this paper the properties considered in [1] by generalizing our proof techniques. We show that a few properties remain the same and, consequently, are not related to the CPU speeds (which is rather counter-intuitive). We give more general results for the remaining cases. We also provide an analysis for additional distributions (e.g., uniform and erlang).

2 Background

2.1 Model of Computation

In this paper, we will consider a quite simple but sufficiently realistic model of computational grid. Our analysis is mainly focused on computational grids, as we assume that network latencies and transfer times are negligible and, thus, do not take into account the data localization. In the grid we consider, there is a central *Resource Broker* (RB), to which each *Computing Element* (CE) is connected, and each client sends its jobs to that central RB. Without lack of generality, we use only one client, but several kinds of workloads have been simulated. Each CE $\mathbb{C}_i$ has a *relative speed* s_i, which is the relative speed of its CPUs compared to a reference CPU. That means that if we have two CPUs with relative speeds s_1 and s_2, a job which takes ℓ units of time on the first CPU to be processed would take $\ell \cdot \frac{s_1}{s_2}$ on the second one. We assume that our system is locally homogeneous, meaning that each CPU of a computing element has the same speed.

Each job j submitted on our system has mainly one parameter: a virtual length (or virtual execution time) ℓ_j, that is, the time that would be taken by a reference CPU (with a relative speed equal to 1) for processing that job. On a CE with relative speed s, the job j will therefore use one CPU during ℓ_j/s units of time. ℓ_j/s is the effective length (or execution time) of j on a CE with relative speed s. We assume that, on one processor, we do not use parallelism or preemption (and then migration), and that our system is greedy.[2]

2.2 Mathematical Model

In the next sections, we will use the following notations: Our system is composed of $\mathcal{N}$ Computing Elements, called $\mathbb{C}_i (i \in [1 \ldots \mathcal{N}])$. c_i refers to the number of CPUs, of $\mathbb{C}_i$, and s_i to the relative speed of $\mathbb{C}_i$. $c_i \cdot s_i$ is the virtual number of CPUs on $\mathbb{C}_i$. C, the total number of virtual CPUs is defined as $C \triangleq \sum_{i=1}^{\mathcal{N}} c_i \cdot s_i$. We define the system load $\nu(t)$ as the total amount of virtual work received in $[0, t]$ divided by the product of the total number of virtual CPUs (C) and the total duration (t). In other words, $\nu(t)$ is the total amount of (virtual) work received divided by the total amount of (virtual) work that the system could provide. Formally,

$$\nu(t) \triangleq \frac{\sum_{\{j \in \mathcal{J} | a_j \leq t\}} \ell_j}{C \cdot t},$$

where $\mathcal{J}$ is the set of jobs submitted to the system and a_j is the job j's submission time. The interval $[0, t]$ is called the *observation period*.

We introduce here a notation: $f_1(t) \sim_t f_2(t)$ means that $\lim_{t\to\infty} \frac{f_1(t)}{f_2(t)} = 1$. Informally, we say that "$f_1(t)$ behaves asymptotically like $f_2(t)$." Notice that $f_1(t) \sim_t f_2(t) \Leftrightarrow f_1(t) = f_2(t) + \varepsilon(t)$, with $\lim_{t\to\infty} \frac{\varepsilon(t)}{f_2(t)} = 0$.

We assume that the arrival of jobs is a random process with an average delay λ^{-1} between two successive arrivals (e.g., the arrivals could be a Poisson process with rate λ, that is, the delay between two successive arrivals follows an Exponential law with the same rate of change). We also assume that the average virtual execution time ($\mathbb{E}[\ell_j]$) has a distribution with mean μ^{-1} (for instance, an Exponential distribution with rate μ). We assume that jobs are independent (i.e., they do not share common resources except CPUs, and there is no precedence constraints between jobs).

2. A system is said to be *greedy* (sometimes called *expedient*) if it never leaves any resource idle intentionally. If a system is greedy, a resource is idle only if there is no eligible job waiting for that resource.

2.3 Random Brokering

2.3.1 Dispatching the Jobs

In this work, we will study a particular case of brokering, where jobs are randomly dispatched, but with a distribution of probability based on the capacity of the different CEs.

The system we analyze is modeled as follows: Each job requires only one CPU, and the broker chooses one among all computing elements in such way that each $\mathbb{C}_i$ has a probability equal to $\frac{c_i \cdot s_i}{C}$ to be chosen. The brokering is then a Bernouilli process, with those probabilities.

It is easy to see (see, for instance, [1]) that, if interarrivals are independent, for each $\mathbb{C}_i$, the interarrival mean will tend toward λ_i^{-1}, where $\lambda_i \triangleq \lambda \frac{c_i \cdot s_i}{C}$, and the average execution time will be $\mu_i^{-1} \triangleq (\mu \cdot s_i)^{-1}$.

2.3.2 Grid Model

We now have all the information we need for defining formally what we consider as being a random brokered grid: A random brokered grid $\mathcal{G}$ is a system $\{\{\mathbb{C}_i\}_{i\in[1..\mathcal{N}]}, \lambda, \mu, R\}$ in which $\{\mathbb{C}_i\}_{i\in[1..\mathcal{N}]}$ is a set of $\mathcal{N}$ computing elements, λ is the inverse of the interarrival averages, μ is the inverse of the average exectution lengths, and R is the used policy, which is the Bernouilli process we defined above.

2.3.3 System Load

Here, we define ν_i as being $\frac{\lambda_i}{\mu_i c_i}$. Then, we have $\nu_i = \frac{\lambda \frac{c_i \cdot s_i}{C}}{\mu s_i c_i} = \frac{\lambda}{\mu C} \triangleq \nu$.

It is not difficult to see that, on average, $\nu(t) \sim_t \nu$; indeed (by the Law of Large Numbers),

$$\begin{array}{lll} \nu(t) & \triangleq \frac{\sum_{\{j\in\mathcal{J}|a_j\le t\}} \ell_j}{C\cdot t} & = \frac{\|\{j\in\mathcal{J}|a_j\le t\}\| \frac{\sum_{\{j\in\mathcal{J}|a_j\le t\}} \ell_j}{\|\{j\in\mathcal{J}|a_j\le t\}\|}}{C\cdot t} \\ & \sim_t \frac{\|\{j\in\mathcal{J}|a_j\le t\}\|}{t}\frac{\mu^{-1}}{C} & \sim_t \frac{\mu^{-1}}{C\cdot\lambda^{-1}} = \nu. \end{array}$$

ν will then be called the system load. If $\nu < 1$, the system will be said *nonsaturated*, and if $\nu > 1$, the system will be considered as *saturated*. The case $\nu = 1$ will not be analyzed in this paper.

In the next two sections, we will first look at the queue size. We then shortly take a glance at two other metrics (number of used CPUs and resorption time) and, finally, we analyze the average slowdown. From a theoretical point of view, the pure saturated case ($\nu > 1$) is not really interesting, because queues grow indefinitely and, therefore, the system "explodes." However, in real systems, the load is often time dependent, and an alternation of saturated and nonsaturated phases can be observed. We are then interested by how queues are growing during a saturated phase, and how queues are decreasing when the load goes down to a nonsaturated phase.

The two cases $\nu < 1$ and $\nu > 1$ require a different approach for their analysis. Thus, the following sections are split in two parts.

3 Queue Size

As a first metric, we analyze the average queue size of computing elements. This metric is useful, for instance, for tuning the local scheduler or for dimensioning the memory needs for buffers.

3.1 $\nu < 1$

Let us first have a look at the case $\nu < 1$. We focus here on a single (arbitrary) $\mathbb{C}_i$, where the arrival is a Poisson process with rate λ_i and the execution time has an Exponential distribution with mean $\mu_i{}^{-1}$. Such a system is well-known, and has been abundantly studied in the literature: This is a $M/M/c_i$ queuing system.

Let J_i be the number of jobs in $\mathbb{C}_i$ (running and waiting jobs), and $\mathbb{P}[J_i = n]$ be the probability that the number of jobs in $\mathbb{C}_i$ is n; we know (see, for instance, [13, p. 371, Section 8.5.2]) that

$$\begin{array}{l} \mathbb{P}[J_i = n] \sim_t b \begin{cases} \frac{(\nu c_i)^n}{n!} & \text{if } n \in [0, c_i] \\ \frac{\nu^n \cdot c_i{}^{c_i}}{c_i!} & \text{otherwise,} \end{cases} \\ \text{where } b \triangleq \left[\sum_{k=0}^{c_i-1} \frac{(\nu c_i)^k}{k!} + \frac{(\nu c_i)^{c_i}}{c_i!}\frac{1}{1-\nu}\right]^{-1}. \end{array} \quad (1)$$

If Q_i is the asymptotic queue size, we have

$$\mathbb{P}[Q_i = n] = \begin{cases} \mathbb{P}[J_i = n + c_i] & \text{if } n > 0 \\ \sum_{k=0}^{c_i} \mathbb{P}[J_i = k] & \text{if } n = 0. \end{cases}$$

Indeed, if there are $n > 0$ jobs in the queue, that means that those n jobs cannot be processed currently, because all CPUs are busy. Therefore, at that time, there are n (in the queue) + c_i (currently processed) jobs in $\mathbb{C}_i$. If there are no jobs in the queue, the probability of this event is equal to $\mathbb{P}[J_i \le c_i]$. Hence,

$$\mathbb{P}[Q_i = n] \sim_t \begin{cases} b\frac{\nu^{n+c_i} \cdot c_i{}^{c_i}}{c_i!} & \text{if } n > 0 \\ \sum_{k=0}^{c_i} b\frac{(\nu c_i)^k}{k!} & \text{if } n = 0. \end{cases}$$

It is now easy to compute the average queue size.

Theorem 3.1. *In a grid $\mathcal{G}$, where $\nu < 1$, arrivals follow a Poisson process and execution times are exponentially distributed, so the average queue size of $\mathbb{C}_i$ is*

$$\begin{array}{l} \mathbb{E}[Q_i] \sim_t b\frac{\nu^{c_i+1} \cdot c_i^{c_i}}{c_i!(1-\nu)^2}, \text{ where} \\ b \triangleq \left[\sum_{k=0}^{c_i-1} \frac{(\nu c_i)^k}{k!} + \frac{(\nu c_i)^{c_i}}{c_i!}\frac{1}{1-\nu}\right]^{-1}. \end{array}$$

Proof.

$$\begin{array}{lll} \mathbb{E}[Q_i] & \sim_t \sum_{n=1}^{\infty} n \cdot b\frac{\nu^{n+c_i} \cdot c_i{}^{c_i}}{c_i!} & \text{by definition of } \mathbb{E} \\ & = b\frac{(\nu c_i)^{c_i}}{c_i!}\sum_{n=1}^{\infty} n\nu^n & \\ & = b\frac{(\nu c_i)^{c_i}}{c_i!}\frac{\nu}{(1-\nu)^2}. & \end{array}$$

Then,

$$\mathbb{E}[Q_i] \sim_t b\frac{\nu^{c_i+1} \cdot c_i^{c_i}}{c_i!(1-\nu)^2}$$

with the same b as above. □

Let us remark that the relative speed s_i of $\mathbb{C}_i$ does not appear in $\mathbb{E}[Q_i]$ when $\nu < 1$. The average queue is then "relative speed independent" in the nonsaturated case.

3.2 $\nu > 1$

When $\nu > 1$, we can free the constraint that we are only looking at Poissonnian systems. The only condition we need is that interarrivals and execution times have a finite average.

We are interested in the $\mathbb{C}_i$ queue size. Obviously, we cannot use the same technique as above, because asymptotically, the queue of a saturated CE is always infinite, regardless of the load, provided that $\nu > 1$. We will focus on the average queue size at time t, that is, if $P_n(t)$ is the probability that there are n jobs in the queue at time t, we will consider $\sum_{n=0}^{\infty} nP_n(t)$.

Before estimating the queue size, we will need to know the number of jobs that have arrived before time t, as well as the number of jobs having left the system by time t.

3.2.1 Arrivals

Let $A_i(t)$ be the random variable which gives the number of jobs submitted between 0 and t (included) to $\mathbb{C}_i$, and $D_i(t)$ the random variable which gives the number of jobs having left $\mathbb{C}_i$ up to t. We have:

Lemma 3.1. *If the arrival process has λ_i^{-1} as interarrival mean and a finite variance, then*

$$\mathbb{E}[A_i(t)] \sim_t \lambda_i t.$$

Proof. It is known (see [4, p. 372, chapter XI, Section 6]) that the number of jobs arriving between 0 and t tends asymptotically (in terms of t) toward a Gaussian with mean $\lambda_i t$ and variance $tv\lambda_i^3$, where v is the variance of interarrivals, which has to be finite. This is an extension of the central limit theorem. We have, therefore, that $\mathbb{E}[A_i(t)]$ behaves asymptotically as $\lambda_i t$. □

However, in practice, such a process converges quite quickly toward a Gaussian; a few dozen iterations are generally sufficient to have a really good approximation, for most distributions encountered in the kind of system we simulated.

3.2.2 Departures

We will now show that the asymptotic behavior of $\mathbb{E}[D_i(t)]$ on t, when $\nu > 1$ is equal to $\mu_i t c_i$. We first need to recall a result from queuing theory: If $\nu > 1$ in a $G/G/c$ queue, the average time for a queue to become empty, when it is not empty, is infinite. Moreover, the average length of an idle period[3] is finite. This is due to the fact that the state "empty queue" is transient; the number of visits to such a state is then (almost surely) finite, and the number of busy periods (between two successive idle periods) is finite. Therefore, the average length of a busy period is infinite, and the average length of an idle period is finite.

We have from this that if $I(t)$ is the sum of each idle period duration on $[0, t]$, then $(t - I(t)) \sim_t t$.

Let $D_i^k(t)$ be the number of jobs sent on $\mathbb{C}_i$, processed by the CPU k, and having left the system by t, let $D^{*k}_i(t)$ be the number of departures before t in a system where we compact the time line in order to remove every idle period, and let $\delta_k(t)$ be the total idle time on CPU k between 0 and t. We have:

Lemma 3.2. $D_i^k(t) = D^{*k}_i(t - \delta_k(t))$, *and* $t - \delta_k(t) \sim_t t$.

Proof. We will bring some changes to our system, in order to remove every idle period. By definition of $D_i^k(t)$, we have that $D_i(t) = \sum_{k=1}^{c_i} D_i^k(t)$, and then

$$\mathbb{E}[D_i(t)] = \mathbb{E}\left[\sum_{k=1}^{c_i} D_i^k(t)\right] = \sum_{k=1}^{c_i} \mathbb{E}[D_i^k(t)]. \tag{2}$$

Let us now look at one specific processor (let's say k). If we observe the usage of that CPU, we can see a succession of idle and busy periods. Let i_1 be the first idle period (if such a period exists), and let d_1 be its duration. We can see that if we move down by a time d_1, the arrival time of each job submitted after i_1 on this CPU, i_1 disappears, but we have that $D_i^k(t) = D'^k_i(t - d_1)$, where D' is the number of jobs leaving the "new" system. If we continue the same process until we have removed all idle periods, we have

$$D_i^k(t) = D^{*k}_i\left(t - \sum_{\ell} d_\ell\right),$$

where there are no more idle periods in the system described by D^* and, by definition, $\delta_k(t) = \sum_\ell d_\ell$. Clearly, $\delta_k(t) \leq I(t)$ (each idle period of the kth CPU is counted in $I(t)$, but some idle periods from other CPUs are in $I(t)$ as well), and then $t - \delta_k(t) \sim_t t$. □

Lemma 3.3. $\mathbb{E}[D_i^k(t)|$ *no idle period in* $[0, t]] \sim_t \mu_i t$.

Proof. If we do not have idle periods in $[0, t]$, the "departure events" is a stochastic process, where interarrival events are distributed as effective job lengths. For instance, if the job lengths distribution is exponential, the departures will be distributed as a Poisson process. We can then use the same reasoning as for Lemma 3.1, using ${\mu_i}^{-1}$ as interarrival mean. □

We now have what we need for estimating the number of departures before t:

Lemma 3.4. $\mathbb{E}[D_i(t)] \sim_t \mu_i t c_i$.

Proof.

$$\begin{aligned}
\mathbb{E}[D_i(t)] &= \textstyle\sum_{k=1}^{c_i} \mathbb{E}[D_i^k(t)] && \text{by Equation (2)}\\
&= \textstyle\sum_{k=1}^{c_i} \mathbb{E}[D^{*k}_i(t - \delta_k(t))] && \text{by Lemma 3.2}\\
&\sim_t \textstyle\sum_{k=1}^{c_i} \mu_i(t - \delta_k(t)) && \text{by Lemma 3.3}\\
&\sim_t \textstyle\sum_{k=1}^{c_i} \mu_i t && \text{because } t - \delta_k(t) \sim_t t\\
& && \text{(Lemma 3.2)}\\
&= \mu_i t c_i.
\end{aligned}$$

□

3.2.3 Number of Jobs in the System

Now, we have the asymptotic behavior of A_i and D_i; we can therefore find the asymptotic behavior of Q_i.

3. An idle period on a CE is a period during which at least one CPU is not running a job and which cannot be extended without containing a period where each CPU is busy. On a single processor, an idle period is a period during which the CPU is not running a job and that cannot be extended.

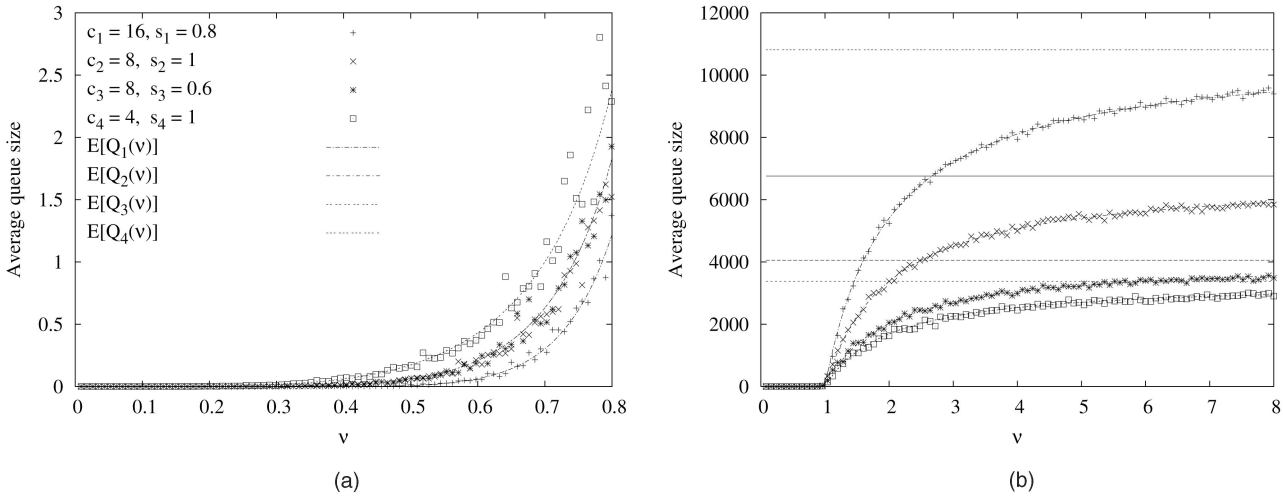


Fig. 1. Queue Size observed in simulation and predicted, for nonsaturated systems ((a)—Theorem 3.1) and saturated systems ((b)—Theorem 3.2). In (b), asymptotes are shown.

Theorem 3.2. *In a grid $\mathcal{G}$ where $\nu > 1$, we have*

$$\mathbb{E}[Q_i(t)] \sim_t \lambda_i t\left(\frac{\nu-1}{\nu}\right).$$

Proof. By definition of J ($J_i(t)$ is the number of jobs in the system—running and waiting—at time t), A and D, we have

$$\begin{array}{lll} J_i(t) & = A_i(t) - D_i(t) & \\ \mathbb{E}[J_i(t)] & = \mathbb{E}[A_i(t)] - \mathbb{E}[D_i(t)] & \\ & \sim_t \lambda_i t - \mu_i t c_i & \text{By Lemmas 3.1 and 3.4} \\ & = t(\lambda_i - \mu_i c_i). & \end{array}$$

Of course, the queue size $Q_i(t)$ is equal to $\max(J_i(t) - c_i, 0)$ and, therefore, in the case we are looking at (saturated and greedy system), we almost always have $Q_i(t) = J_i(t) - c_i$. Therefore,

$$\begin{array}{lll} \mathbb{E}[Q_i(t)] & = \mathbb{E}[J_i(t)] - \epsilon(c_i) & \text{where } \epsilon(c_i) \text{ is an integer in } [0..c_i] \\ & \sim_t t(\lambda_i - \mu_i c_i) - \epsilon(c_i) & \\ & \sim_t t(\lambda_i - \mu_i c_i) & \text{Because } c_i \text{ is constant with regard to } t \\ & = \lambda_i t \frac{\nu-1}{\nu}. & \end{array}$$

□

The slope of the queue growth is therefore $(\lambda_i - \mu_i c_i)$.

We can now easily calculate the total number of jobs being in a queue of the whole system; we just need to sum up those queue sizes:

$$\mathbb{E}[Q(t)] = \sum_i \lambda_i t \frac{\nu-1}{\nu} = \lambda t \frac{\nu-1}{\nu}.$$

3.3 Experimental Results

In this paper, we are interested in various metrics, like the (average) queue size, the (average) slowdown, the (average) number of used CPUs... and, in particular, in their asymptotic behavior for various system loads. In order to illustrate that, we will plot those metrics (y-axis), observed by simulation or predicted by theoretical analysis, obtained for a (large) fixed time t, as a function of the system load ν (x-axis). For instance, in Fig. 1, we performed about 100 simulations for various νs going from 0 to 8 (0 to 0.8 for Fig. 1a), and noticed the average queue size after a "long" constant time for each computing element (four dots vertically aligned); we also plotted in the same figure the values predicted by the theory (lines). The size (number of CPUs and the relative speeds) of the simulated CE is shown in the legend.

In the examples we picked from our simulations, we used the following parameters: the simulation duration $t = 100,000$; the average interarrival delay $\frac{1}{\lambda} = 2$; the four CEs have 16, 8, 8, and 4 CPUs (c_i); and the relative speeds are, respectively, 0.8, 1, 0.6, and 1.

In the first plot (Fig. 1a), we put our results for $\nu < 1$ (using Theorem 3.1, which is only valid for exponential distributions), and in the second plot, we show the case $\nu > 1$ (using Theorem 3.2). We did not put these two parts on the same plot simply for readability: The order of magnitude for the first part is around 10 (queue sizes are lower than 10 up to $\nu \simeq 0.98$), and around 10,000 for the second part (the biggest queue size tends asymptotically toward $\sim$ 10,810).

We observe that there is an "inversion" around $\nu = 1$: When $\nu < 1$, $\mathbb{E}[Q_i] < \mathbb{E}[Q_j]$ if $c_i > c_j$, and, for $\nu > 1$, we have the opposite. More precisely, we have that $\mathbb{E}[Q_i] > \mathbb{E}[Q_j]$ if $\lambda_i > \lambda_j$ or if $c_i s_i > c_j s_j$. We can see as well that, as we mentioned above, in the nonsaturated case, queue size does not depend on relative speed. We can see in Fig. 1a that $\mathbb{C}_2$ and $\mathbb{C}_3$ have the same average, in spite of the fact that $\mathbb{C}_2$ is almost twice as powerful than $\mathbb{C}_3$.

If, in the saturated case, a CE with c CPUs with relative speed s is equivalent to a CE with c/n CPUs with relative speed $s \cdot n$ (with $c/n \in \mathbb{N}$), this is not the case in nonsaturated case: Two CEs with the same number of CPUs are "equivalent" (in terms of queue size), whatever the relative speeds.

In order to give a more accurate view on the precision of our estimation, we show two more logarithmic plots in Fig. 2, giving the ratio between observations and theoretical

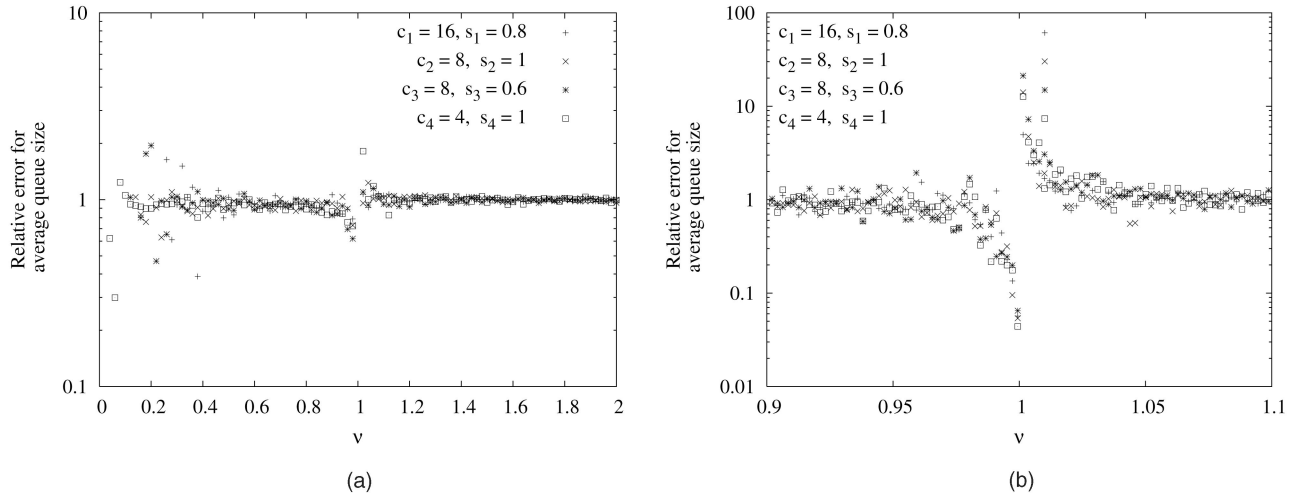


Fig. 2. Relative estimation error (log scale).

predictions. The first plot shows the ratio for ν varying between 0 and 2, and the second one focuses on a short interval around 1. In $0-2$, we can divide our observations into three parts: $\nu \in [0, 0.5]$, $\nu \in [0.5, \sim 0.95]$, and $\nu \in [\sim 1.05, \rightarrow [$ (see below for the missing interval). In the first part, we have that 40 percent of observations were in the interval $[p \pm 10\%]$ where p is the theoretical prediction, and 60 percent in $[p \pm 20\%]$. When ν was in $[0.5, 0.95]$, results were much more precise: 77 percent in $[p \pm 10\%]$ and 100 percent in $[p \pm 20\%]$. After 1.05, we had some even better results: 98 percent in $[p \pm 10\%]$, and 100 percent in $[p \pm 20\%]$.

Bad results close to $\nu = 0$ are mainly due to the fact that the predicted queue size is always > 0 (but really small), while we often observed an empty queue. This results in 100 percent errors. If we do not take thoses observations into account, the $[p \pm 10\%]$ interval goes up to 53 percent and the next one to 76 percent.

The second plot of Fig. 2 highlights the limits of our models: for $\nu < 0$, we assume that the observation period was infinite, which is of course not the case. Our predictions overestimate the reality. For $\nu > 1$, we assume that the queues were never empty, which does not happen for loads close to 1. In this case, predictions underestimate the reality. However, we notice that our estimations are far from observation only for a very small interval around 1 ($\sim [0.97, 1.03]$ for the example we presented).

4 Used CPUs

The average number of used CPUs corresponds to the average number of jobs running on a Computing Element in $[0, t]$. In the case of sequential jobs, a simple argument allows us to find this average, in the saturated case and in the nonsaturated case. The first case is trivial: If the system is saturated and queues are almost never empty, each CPU is continuously processing a job. The average number of used CPUs on $\mathbb{C}_i$ is therefore c_i for any $\nu > 1$. For the second case, we know (see Lemma 3.1) that $\mathbb{E}[A_i(t)] \sim_t \lambda_i t$. Let's assume that at time t, the queue is empty, or at least with a size negligible regarding the $A_i(t)$. The (average) total amount of virtual work processed in $[0, t]$ tends, therefore, toward $\frac{\lambda_i t}{\mu}$ (the number of jobs multiplied by the average length). Thus, on average, $\frac{\lambda_i}{\mu}$ virtual CPUs, or $\frac{\lambda_i}{\mu s_i}$ real (or effective) CPUs are needed, which is equal to νc_i.

As for queue size, we can easily obtain the total number of used CPUs (for $\nu > 1$): $\sum_i \nu c_i = \nu \sum_i c_i$.

Fig. 3 confirms the validity of our argument. We used the same grid configuration as for Fig. 1.

5 Resorption Time

From a theoretical point of view, there are no obvious reasons to consider a system where $\nu > 1$. Indeed, this kind of system is not stable, and its asymptotic behavior is not viable. However, we find this analysis interesting because it allows us to answer the following question: If, starting from a point where all queues are (almost) empty, the load of our system is $\nu_1 > 1$ during a period t_1, and, after that period, the load goes down to $\nu_2 < 1$, how long will it take before all queues are (almost) completely resorbed? We are not providing a precise proof, but rather a rough estimation of that resorption time.

We showed in Section 4 that if the load is $\nu < 1$, there is, on average, $c_i \nu$ (real) used CPUs, and then $c_i(1 - \nu)$ free CPUs on $\mathbb{C}_i$. We know as well from Lemma 3.4 that, if we

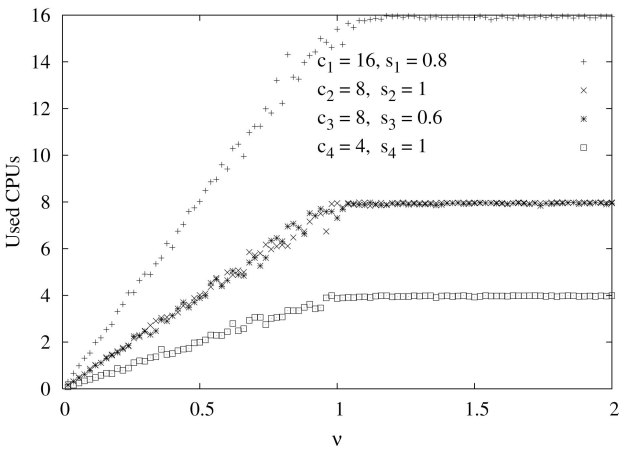


Fig. 3. Average CPU usage.

have k CPUs, $\mathbb{C}_i$ takes, on average, $\frac{N_i}{k\mu s_i}$ units of time for processing N_i jobs with average virtual length μ. We then consider the following approximation giving the time needed by $\mathbb{C}_i$ for resorbing N_i sequential jobs with average virtual length μ', if the load is ν:

$$\frac{N_i}{c_i(1-\nu)\mu' s_i}.$$

As a direct corollary of Theorem 3.2, we show that if there are N jobs waiting in the whole system, $\frac{Nc_is_i}{C}$ are, on average, in $\mathbb{C}_i$. The resorption no longer depends on i, and we have:

Theorem 5.1. *The mean time needed for resorbing N sequential jobs (fairly distributed, with average length μ') in a grid $\mathcal{G}$ where $\nu < 1$ is approximately*

$$\frac{N}{C(1-\nu)\mu'}.$$

In particular, these results mean that if the input is suddenly stopped after having a load $\nu' > 1$ during t units of time ($N = \lambda t \frac{\nu'-1}{\nu'}$), the resorption time would be

$$\frac{\lambda t \frac{\nu'-1}{\nu'}}{C\mu'} = t(\nu' - 1).$$

6 Slowdown

The slowdown for a particular job is classically defined as the ratio of a task's perceived exectution time to its execution time in an unloaded environment, or, formally,

$$\frac{\text{waiting time} + \text{execution time}}{\text{execution time}}.$$

In our case, we need to slightly modify that ratio, due to the heterogeneity of our system. We mainly want the same job with the same completion time (waiting time + effective execution time) on two CEs with different relative speeds having the same slowdown on those CEs. We assume that the user does not care that the real execution time of his job was shorter or longer than he thought, he only cares about the time he waits before getting back his results.

We will then define the slowdown as (for a formal definition of effective and virtual execution time, cf. Section 2.1):

$$\frac{\text{waiting time} + \text{effective execution time}}{\text{virtual execution time}}.$$

If the average waiting time—that is, the time between the submission time and the beginning of the job execution—for a job submitted on $\mathbb{C}_i$ at time θ with a load ν is $W_i(\theta,\nu)$, the average slowdown for a job submitted on $\mathbb{C}_i$ at time θ with a load ν will be ($f(m)$ is the probability density for virtual job length)

$$\begin{aligned}\mathbb{E}[\mathcal{SD}_i(\theta)] &= \int_0^\infty \frac{W_i(\theta,\nu) + m/s_i}{m} f(m)dm \\ &= W_i(\theta,\nu)\int_0^\infty \frac{f(m)}{m}dm + {s_i}^{-1}\end{aligned} \tag{3}$$

because $\int_0^\infty f(m)dm = 1$, by definition of the probability density function. This average is valid only if waiting times are independent of the execution times, which is true for simple scheduling strategies such as FCFS or FF, but is no longer true for more complex techniques such as Backfilling [12].

If that function is not continuous, and that job length can take the values m_n, $n \in [1 \dots N]$, with a probability $P(m_n)$, the slowdown will be

$$\mathbb{E}[\mathcal{SD}_i(\theta)] = W_i(\theta,\nu)\sum_{n=1}^{N}\frac{P(m_n)}{m_n} + {s_i}^{-1}.$$

6.1 Preliminary

Before splitting our argument in two cases ($\nu < 1$ and $\nu > 1$), we compute the integral (or summation) in particular cases, keeping in mind first that $f(m)$ is a probability density function ($\int_0^\infty f(m)dm = 1$) and, second, that its average is μ ($\int_0^\infty mf(m)dm = \mu$).

We use several distributions for the virtual execution time:

- Constant: Each job has a constant virtual execution time of μ^{-1}.
- Uniform: Virtual execution times have a uniform distribution going from $(1-\alpha)/\mu$ to $(1+\alpha)/\mu$ for α in $]0,1[$.
- Shifted exponential: As it is unfortunately not really possible to compute the average slowdown for an exponential distribution—the non-null probability density to have a zero length job makes the integral infinite—we will slightly modify the exponential distribution in the following way: The job length is a constant $\frac{\alpha}{\mu}$ plus an exponential distribution with a rate $\frac{\mu}{1-\alpha}$, with $\alpha \in]0,1[$.
- Erlang: Virtual execution times have a one-dimensional Erlang distribution with a shape h. See [11, p. 72, equation 2.147] or [13, p. 153, equation 4.87] for more details.

Table 1 summarizes the value of $\frac{\int_0^\infty f(m)}{m}$ for those distributions. The incomplete Euler Gamma function $\Gamma[a,z]$ is defined as being $\int_z^\infty t^{a-1}e^{-t}dt$.

It can be observed that, in many cases, the value of $\int_0^\infty \frac{f(m)}{m}dm$ is inversely linear in the average, meaning in the form μA, where A is a constant depending of the distribution parameters, but not μ. It has been checked for other distributions as well, such as Weibull or LogNormal.

6.2 $\nu < 1$

Knowing the average waiting time (or the average queue size) is required in order to estimate the slowdown. Unfortunately, in the nonsaturated case, we only have results for the Poissonnian case. Our results for the average slowdown are then also limited.

Theorem 6.1. *In a grid $\mathcal{G}$ where $\nu < 1$, with shifted exponential distribution for job length, the average slowdown $\mathbb{E}[\mathcal{SD}_i]$ is asymptotically close to*

$$\frac{1}{c_is_i}\,\frac{b(\nu c_i)^{c_i}}{c_i!(1-\nu)^2}\,\frac{e^{\frac{\alpha}{1-\alpha}}}{1-\alpha}\,\Gamma[0,\frac{\alpha}{1-\alpha}] + {s_i}^{-1},$$

where b is the one defined in Theorem 3.1.

TABLE 1
Summary of Distributions Used in This Paper

Distribution	$f(m)$	$\int_0^\infty \frac{f(m)}{m} dm$
Constant	$N = 1, m_1 = \mu^{-1}, \mathbb{P}(m_1) = 1$	μ
Uniform	$f(m) = \begin{cases} \frac{\mu}{2\alpha} & \text{if } \frac{1-\alpha}{\mu} < m < \frac{1+\alpha}{\mu} \\ 0 & \text{otherwise} \end{cases}$	$\frac{\mu}{2\alpha} \log \frac{1+\alpha}{1-\alpha}$
Shifted exponential	$f(m) = \begin{cases} 0 & \text{if } m < \frac{\alpha}{\mu} \\ \frac{\mu}{1-\alpha} e^{-(m-\frac{\alpha}{\mu})\frac{\mu}{1-\alpha}} & \text{otherwise} \end{cases}$	$\frac{\mu}{1-\alpha} e^{\frac{\alpha}{1-\alpha}} \Gamma[0, \frac{\alpha}{1-\alpha}]$
Erlang	$f(m) = \frac{(\mu h)^h}{(h-1)!} m^{h-1} e^{-\mu m h}$	$\mu \frac{h}{h-1}$

Proof. As in Section 3.1, for $\nu < 1$, we first consider the Poisson-Exponential case.

We know by (3) that $\mathbb{E}[\mathcal{SD}_i(t)] = W_i(\theta, \nu) \int_0^\infty \frac{f(m)}{m} dm + {s_i}^{-1}$. In the Poissonnian case, we can compute $W_i(\theta, \nu)$.

When there is at least one free CPU on $\mathbb{C}_i$, that is, when J_i is in $[0, c_i - 1]$, the waiting time is null. If each CPU is processing some work and there are n (possibly 0) jobs in the queue, the arriving job will in the average wait $\frac{n+1}{\mu c_i} = \frac{\nu}{\lambda_i}(n+1)$ (because execution times are exponentially distributed). The average waiting time is then

$$W_i(\theta,\nu) = \underbrace{\sum_{n=0}^{c_i-1} 0 \cdot \mathbb{P}[J_i = n]}_{=0} + \sum_{n=c_i}^{\infty} \frac{\nu}{\lambda_i}(n - c_i + 1)\mathbb{P}[J_i = n]$$

$$= \frac{\nu}{\lambda_i} \underbrace{\sum_{q=0}^{\infty} q\mathbb{P}[J_i = q + c]}_{=\mathbb{E}[Q_i]} + \frac{\nu}{\lambda_i} \underbrace{\sum_{q=0}^{\infty} \mathbb{P}[J_i = q + c_i]}_{=A}$$

with (by (1))

$$A = \sum_{q=0}^{\infty} \mathbb{P}[J_i = q + c_i] \sim_\theta \sum_{q=0}^{\infty} b \frac{\nu^{q+c_i} {c_i}^{c_i}}{c_i!} \sim_\theta b \frac{(\nu c_i)^{c_i}}{c_i!} \frac{1}{1-\nu}$$

and (by Theorem 3.1)

$$\mathbb{E}[Q_i] = b \frac{\nu^{c_i+1} {c_i}^{c_i}}{c_i!(1-\nu)^2}.$$

Then,

$$W_i(\nu,\theta) \sim_\theta \frac{\nu}{\lambda_i} \left(b \frac{\nu^{c_i+1} {c_i}^{c_i}}{c_i!(1-\nu)^2} + b \frac{(\nu c_i)^{c_i}}{c_i!} \frac{1}{1-\nu} \right) = \frac{\nu}{\lambda_i} \frac{b(\nu c_i)^{c_i}}{c_i!(1-\nu)^2}.$$

As quoted above, for a "pure" exponential distribution, the integral is infinite. However, our simulations have shown that if we slightly modify our job length distribution (we kept the interarrival distribution and we used a shifted exponential distribution for job lengths with a small α), queue sizes are still quite close to the $\mathbb{E}[Q_i]$ we obtained above. Despite the fact that we cannot give an exact average or the asymptotic behavior of the slowdown, we are going to give an approximation of this average. We consider for that estimation that the $\mathbb{E}[Q_i]$ we got in Theorem 3.1 is still valid for a shifted exponential distribution for job lengths, if α is close enough to 0. We then give the approximation for this average:

$$\mathbb{E}[\mathcal{SD}_i] \simeq \frac{\nu}{\lambda_i} \frac{b(\nu c_i)^{c_i}}{c_i!(1-\nu)^2} \frac{\mu}{1-\alpha} e^{\frac{\alpha}{1-\alpha}} \Gamma\left[0, \frac{\alpha}{1-\alpha}\right] + {s_i}^{-1}$$

$$= \frac{1}{c_i s_i} \frac{b(\nu c_i)^{c_i}}{c_i!(1-\nu)^2} \frac{e^{\frac{\alpha}{1-\alpha}}}{1-\alpha} \Gamma\left[0, \frac{\alpha}{1-\alpha}\right] + {s_i}^{-1},$$

where b is the one defined in Theorem 3.1. □

We can see that, in case of real exponential distribution (that is, shifted distribution with parameter $\alpha = 0$), the average queue size is infinite, because $\Gamma[0, 0]$ is infinite. However, the Γ function grows quite slowly: $\Gamma[0, \frac{\alpha}{1-\alpha}]$ is greater than 100 only for $\alpha < 10^{-43}$ ($\Gamma[0, \frac{\alpha}{1-\alpha}]$ is linear in $-\log(\alpha)$).

6.3 $\nu > 1$

In the system we are studying, we measure the slowdown of completed jobs. We then compute the average for each measured job.[4] But, at the end of our observation period, especially if $\nu \gg 1$, a lot of jobs are still in the queue and are therefore not taken into account in our average. The average slowdown is thus measured between the first job and the last finished job (and not the last submitted job) before the end of our observation period. Of course, the heavier the load, the earlier the last finished job was submitted, and the fewer the number of jobs that are part of the average.

In order to predict the average slowdown between the first job and the last one leaving the system before the end of the observation period, we will proceed in two steps; first, we predict the average slowdown of the job submitted at a time θ ($\mathbb{E}[\mathcal{SD}_i(\theta)]$), then we will use these results for predicting the average slowdown on each job leaving the system between 0 and a time t, or the measured slowdown ($\mathbb{E}[\mathcal{MSD}_i(t)]$). Notice that, if $\nu < 1$, $\mathbb{E}[\mathcal{SD}_i(t)] \sim_t \mathbb{E}[\mathcal{MSD}_i(t)]$.

6.3.1 Slowdown for a Job Submitted at Time θ

Equation (3) gave us the general form of the slowdown for a job submitted at time θ. We computed in the previous section

4. We are not fully assured that there is not a problem in this case: A job has to wait for another job completion and, thus, slowdowns may not be independent. We still need to check that slowdowns are independent and, if not, if this could cause some problems.

the value of the integral part for several job length distributions; we still need to estimate the waiting time $W_i(\theta,\nu)$.

Lemma 6.1. *If $\nu > 1$, the average waiting time for a job submitted at time θ is*

$$W_i(\nu,\theta) \sim_\theta \theta(\nu-1).$$

Proof. From Theorem 3.2, we know that the average number of jobs in queue i is asymptotically (for $\theta \to \infty$)

$$\lambda_i \theta \frac{\nu-1}{\nu}.$$

The average time the N^{th} job in the queue i (c_i jobs are running, $N-1$ jobs are waiting before this job) will wait tends (for $N \to \infty$) to $\frac{N}{\mu_i c_i}$. In order to be convinced of this, let us remark that the job flow at input of CPUs (that is, the flow at the output of queue of jobs coming in the CPUs) is, on average, equal to the job flow at the output of CPUs. We therefore know that, on average, the queue is emptied[5] according to $\mu_i t c_i$ (see Lemma 3.4) and, thus, that the Nth job waits in the queue for an average of $\frac{N}{\mu_i c_i}$ units of time.

Then, the average waiting time for a job coming at time t on $\mathbb{C}_i$ is (because $\frac{N}{\mu_i c_i}$ is linear in N)

$$W_i(\nu,\theta) \sim_t \lambda_i \theta \frac{\nu-1}{\nu}\frac{1}{\mu_i c_i} = \theta(\nu-1)$$

because $\frac{\lambda_i}{\mu_i c_i} = \nu$. □

Let us remark that we found the same result as for the comment we did after Theorem 5.1 (end of Section 5), which is not surprising because the resorption time after a time θ is equivalent to the waiting time of the last job submitted at that time.

And, thus,

$$\begin{aligned}\mathbb{E}[\mathcal{SD}_i(\theta)] &\sim_\theta \theta(\nu-1)\int_0^\infty \frac{f(m)}{m}dm + {s_i}^{-1} \\ &\sim_\theta \theta(\nu-1)\int_0^\infty \frac{f(m)}{m}dm.\end{aligned} \tag{4}$$

The following lemma is simply proved by replacing the integral in (4) by the results we got above (Table 1).

Theorem 6.2. *In a grid $\mathcal{G}$ where $\nu > 1$, the average slowdown of the job submitted at time θ is*

$$\mathbb{E}[\mathcal{SD}_i(\theta)] \sim_\theta \lambda\theta\frac{\nu-1}{\nu C}\cdot\begin{cases}1 & (1)\\ \frac{1}{2\alpha}\log\left(\frac{1+\alpha}{1-\alpha}\right) & (2)\\ e^{\frac{\alpha}{1-\alpha}}\frac{\Gamma[0,\frac{\alpha}{1-\alpha}]}{1-\alpha} & (3)\\ \frac{h}{h-1} & (4)\end{cases}$$

if virtual execution times distribution is, respectively,

1. *constant with value μ^{-1},*
2. *uniform going from $(1-\alpha)/\mu$ to $(1+\alpha)/\mu$ for α in $]0,1[$,*
3. *shifted exponential with parameter α,*
4. *Erlang with parameter h.*

5. That does not mean that the queue size goes down at that speed, but jobs are leaving the queue at that speed.

6.3.2 Average Slowdown Until the Last Finished Job with $\nu > 1$

We know the slowdown of a job submitted at a given time. We will now estimate the submission time of the job finishing at time τ, and compute the average slowdown between 0 and the submission time of the last finished job.

Lemma 6.2. *A job finishing at time τ has been submitted (on average) at time*

$$\theta = \frac{\lambda\tau - \nu C}{\lambda\nu}.$$

Proof. If a job is submitted at a time θ in queue i, it will end up in the average at a time which is the sum of the arrival time, the average queuing time, and the average execution time: $\tau \sim_\theta \theta + \theta(\nu-1) + \mu^{-1} = \nu\theta + \frac{\nu C}{\lambda}$. □

In order to give an estimation of the average slowdown from 0 to $\frac{\lambda s_i t - \nu C}{\lambda s_i \nu}$ (that is, the average slowdown measured at time t), we need to know the job length distribution as above.

Theorem 6.3. *In a grid $\mathcal{G}$ where $\nu > 1$, the average slowdown for jobs leaving the system between 0 and t, in the case of constant execution time μ^{-1}, tends asymptotically on t toward*

$$\lambda t \frac{\nu-1}{2\nu^2 C}.$$

Proof. We use here a reasoning close to the one we used for the estimation of the departure in queue size when $\nu > 1$. We "compact" our jobs in a way that we no longer have idle periods in 0 and $t - \delta_k(t)$. This manipulation does not change the slowdown of any job and, thus, keeps the average slowdown. Because the execution time is constant, we have, on the kth processor in the new system, one job ending at time ${\mu_i}^{-1}$ (or $\frac{\nu C}{\lambda s_i}$), another one ending at time $2{\mu_i}^{-1}\ldots$ and one ending at time $\lfloor \mu_i(t-\delta_k(t))\rfloor {\mu_i}^{-1}$.

We know, furthermore, by Theorem 6.2, (1), and Lemma 6.2, that a job ending at time τ has been slowed down in the average by

$$\frac{\lambda s_i \tau - \nu C}{\lambda s_i \nu}\frac{\lambda(\nu-1)}{C\nu} = \lambda\frac{\tau(\nu-1)}{\nu^2 C} + \frac{1-\nu}{s_i\nu} \sim_\tau \lambda\tau\frac{\nu-1}{\nu^2 C}.$$

If $\mathcal{MSD}_i^k(t)$ is the measured slowdown on the kth CPU of $\mathbb{C}_i$ between 0 and t, and $\mathcal{MSD}^{*k}{}_i(t')$ is the same in the modified system, we then have that (t' denotes $t - \delta_k(t)$)

$$\begin{aligned}\mathbb{E}[\mathcal{MSD}_i^k(t)] &= \mathbb{E}[\mathcal{MSD}_i^{*k}(t')] \\ &\sim_t \tfrac{1}{\lfloor\frac{t'\lambda s_i}{\nu C}\rfloor}\textstyle\sum_{j=1}^{\lfloor\frac{t'\lambda s_i}{\nu C}\rfloor}\left(j\frac{\nu C}{\lambda s_i}\frac{\lambda(\nu-1)}{\nu^2 C}\right) \\ &= \tfrac{1}{\lfloor\frac{t'\lambda s_i}{\nu C}\rfloor}\left(\tfrac{\nu-1}{s_i\nu}\tfrac{\lfloor\frac{t'\lambda s_i}{\nu C}\rfloor(\lfloor\frac{t'\lambda s_i}{\nu C}\rfloor+1)}{2}\right) \\ &= \tfrac{\nu-1}{2s_i\nu}\left(\lfloor\tfrac{t'\lambda s_i}{\nu C}\rfloor+1\right) \\ &\sim_t \tfrac{\nu-1}{2s_i\nu}\left(\lfloor\tfrac{t\lambda s_i}{\nu C}\rfloor+1\right) \qquad \text{because } t' \sim_t t \\ &\sim_t \lambda t\tfrac{\nu-1}{2\nu^2 C}.\end{aligned}$$

We can now compute the average measured slowdown for all our systems $\mathbb{E}[\mathcal{MSD}_i(t)]$:

$$\begin{aligned}\mathbb{E}[\mathcal{MSD}_i(t)] &= \tfrac{1}{c_i}\textstyle\sum_{k=1}^{c_i}\mathbb{E}[\mathcal{MSD}_i^k(t)]\\ &\sim_t \tfrac{1}{c_i}\textstyle\sum_{k=1}^{c_i}\lambda t\tfrac{\nu-1}{2\nu^2C}\\ &= \lambda t\tfrac{\nu-1}{2\nu^2C}.\end{aligned}$$

□

When the execution time is not constant, we admit an approximation of the slowdown. We consider that this time is really small compared to t and sufficiently regular and replace the summation by an integral ($d\tau$ is the execution time).

Theorem 6.4. *In a grid $\mathcal{G}$ where $\nu > 1$, the average slowdown for jobs leaving the system between 0 and t, in the case of uniform execution distribution between $\mu(1-\alpha)$ and $\mu(1+\alpha)$, tends asymptotically on t toward*

$$\lambda t\frac{\nu-1}{2\nu^2C}\cdot\frac{1}{2\alpha}\log\left(\frac{1+\alpha}{1-\alpha}\right).$$

Proof. In the case of uniform distribution, the job ending at time τ has been slowed down by a time

$$\frac{\lambda\tau-\nu C}{\lambda\nu}\lambda\frac{\nu-1}{2\alpha C\nu}\log\left(\frac{1+\alpha}{1-\alpha}\right) = \log\left(\frac{1+\alpha}{1-\alpha}\right)\frac{\nu-1}{2\alpha\nu}\left(\frac{\lambda\tau}{C\nu}-1\right).$$

The average slowdown measured at time t can be approximated by

$$\begin{aligned}&\frac{1}{t'}\int_0^{t'}\log\left(\frac{1+\alpha}{1-\alpha}\right)\frac{\nu-1}{2\alpha\nu}\left(\frac{\lambda\tau}{C\nu}-1\right)d\tau\\ &= \log\left(\frac{1+\alpha}{1-\alpha}\right)\frac{\nu-1}{2\alpha\nu}\left(\frac{\lambda t'}{2C\nu}-1\right)\sim_t \lambda t\frac{\nu-1}{2\nu^2C}\cdot\frac{1}{2\alpha}\log\left(\frac{1+\alpha}{1-\alpha}\right),\end{aligned}$$

where t' is the instant of the last termination before, or just at, $t-\delta_k(t)$.

Because 1 is negligible compared to $\frac{\lambda t'}{2C\nu}$ and $t' = t-\varepsilon-\delta_k(t)\sim_t t$, where ε is lower than the execution time of the first job ending after t, which is, on the average, μ, and does not depend upon t. □

Theorem 6.5. *In a grid $\mathcal{G}$ where $\nu > 1$, the average slowdown for jobs leaving the system between 0 and t, in the case of shifted exponential distribution with parameter α, tends asymptotically on t toward*

$$\lambda t\frac{\nu-1}{2\nu^2C}\cdot e^{\frac{\alpha}{1-\alpha}}\frac{\Gamma[0,\frac{\alpha}{1-\alpha}]}{1-\alpha}.$$

Proof. With the same argument, we found that in the case of a shifted exponential, the job ending at time τ has been slowed down by a time

$$\frac{\lambda\tau-\nu C}{\lambda\nu}\lambda\frac{\nu-1}{\nu C}e^{\frac{\alpha}{1-\alpha}}\frac{\Gamma[0,\frac{\alpha}{1-\alpha}]}{1-\alpha}$$

and that the average slowdown measured at time t tends toward

$$\lambda t\frac{\nu-1}{2\nu^2C}\cdot e^{\frac{\alpha}{1-\alpha}}\frac{\Gamma[0,\frac{\alpha}{1-\alpha}]}{1-\alpha}.$$

□

Theorem 6.6. *In a grid $\mathcal{G}$ where $\nu > 1$, the average slowdown for jobs leaving the system between 0 and t, in the case of Erlang distribution with shape h, tends asymptotically on t toward*

$$\lambda t\frac{\nu-1}{2\nu^2C}\cdot\frac{h}{h-1}.$$

6.4 Experimental Results

As for queue sizes, and for the same readability reasons, we show the cases $\nu < 1$ and $\nu > 1$ separately. In the case $\nu < 1$, we only found results for (nearly) exponential distribution. Fig. 4a shows this case. For $\nu > 1$, we obtained results for different distributions (fixed, uniform, shifted exponential, and Erlang). Our predictions are superimposed to our simulation results in Figs. 4 and 5.

The most interesting difference between $\nu < 1$ and $\nu > 1$ is probably that, in the first case, the average slowdown depends upon the CE on which the concerned job has been submitted, while, when $\nu > 1$, the average slowdown is the same for each CE. This appears in formulas: average slowdown for $\nu < 1$ contains is, and average slowdown for $\nu > 1$ is not "i-dependent." However, we have to notice that we do not compute the average slowdown on the same number of jobs. For the first part, we compute the average on almost all submitted jobs, while, in the second part, the bigger ν is, the more the proportion of measured job is small.

For $\nu < 1$, the slowdown we give corresponds to the average factor a job submitted at any time would be slowed down. For $\nu > 1$, it is not the case; the slowdown for a job submitted at time t would be, on average, the one we gave in Theorem 6.2, but is still "i-dependent."

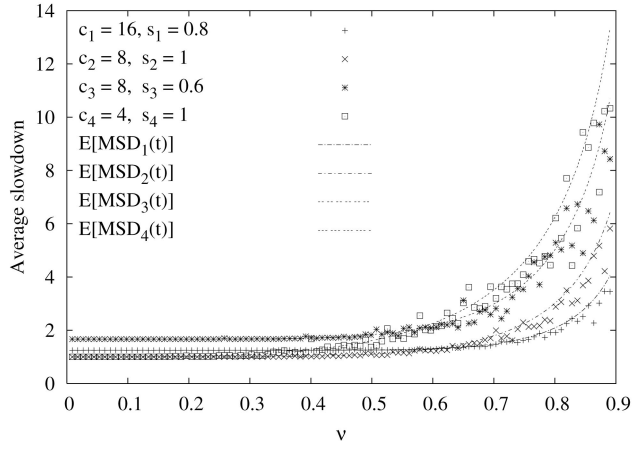
As we could expect, when the job length distribution does not allow small jobs, the observed dispersion is rather small. In the case of shifted exponential distribution with a small α (Fig. 5b), we observe some surprising extreme values; the values come generally from a really small job submitted late in the system. This kind of job waits a very long time in the queue (compared to its execution time) and then has a heavy weight in the average. In a real situation, the same problem occurs when a job crashes at the very beginning of its execution, which distorts the average slowdown. It is why Feitelson et al. [3] propose their *bounded slowdown*, which reduces the weight of small jobs.

Another remark is that the average slowdown (on finished jobs) is notably greater in some distributions than in others, despite the fact that, as we have shown previously, queue sizes do not depend (too much) on job length distribution.

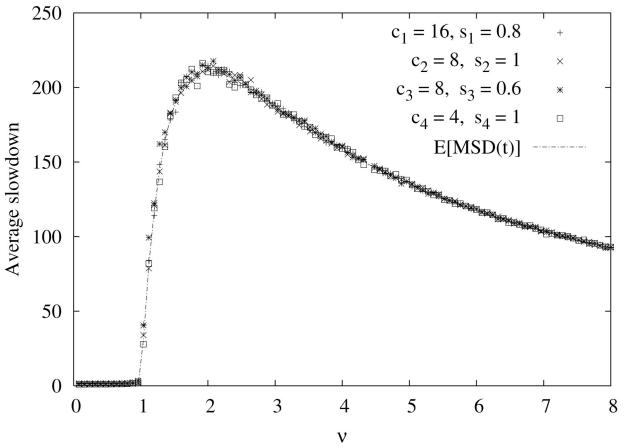
Figs. 4 and 5 show how our predictions are close to our observations. We see that in the case of a shifted exponential, the smaller α is, the bigger the dispersion becomes.

7 Conclusions

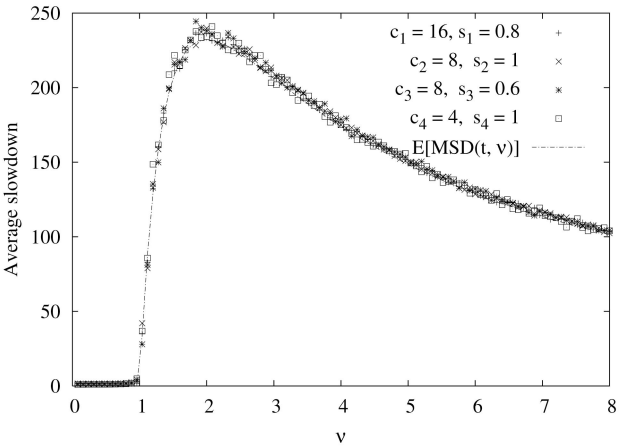
Scheduling is one of the key services required for enabling performance on distributed and heterogeneous platforms, such as computational grids. In this paper, we have focused on a special kind of scheduler, called a resource broker. A resource broker (also called a metascheduler) is mandatory when dealing with a multilevel architecture such as the one provided by the EDG/EGEE infrastructure. A resource



(a)

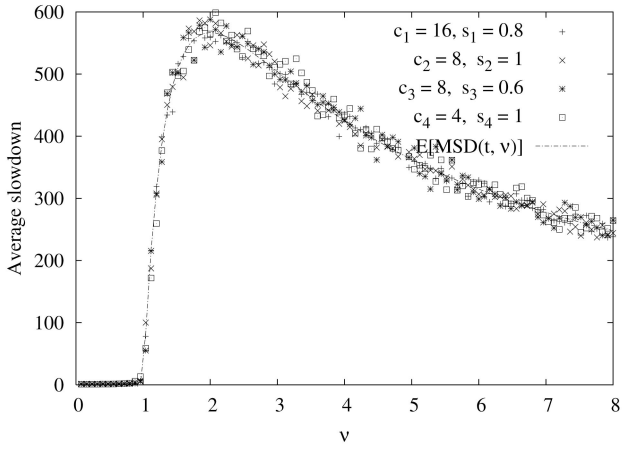


(b)

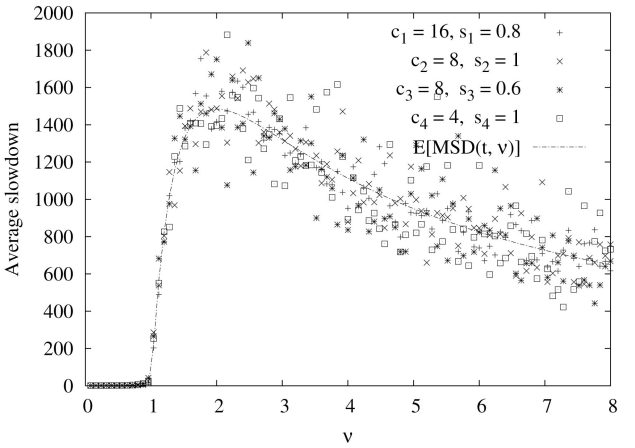


(c)

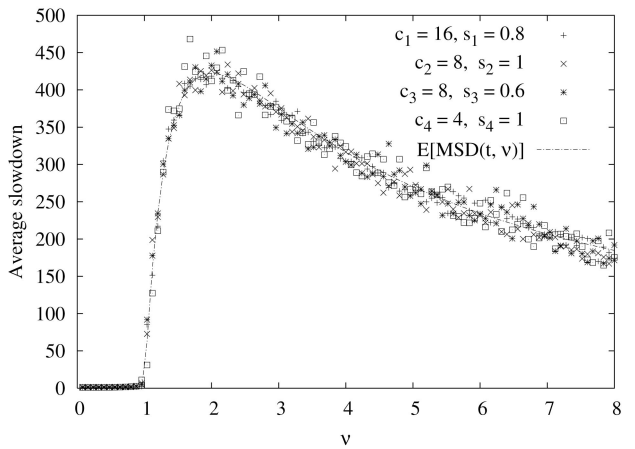
Fig. 4. Average slowdown, with (a) exponential distribution shifted with $\alpha = 0.0005$ on nonsaturated system (Theorem 6.1), (b) fixed job length (Theorem 6.3) on saturated system, and (c) uniform distribution from 0.5μ to 1.5μ on saturated system (Theorem 6.4).



(a)



(b)



(c)

Fig. 5. Average slowdown on saturated systems, with (a) shifted Exponential distribution with $\alpha = 0.05$, (b) idem with $\alpha = 0.0005$ (Theorem 6.5), and (c) Erlang with shape (h)=2.

broker dispatches requests on computing elements and each computing element schedules its own requests on its processors following a given policy (FIFO, etc.).

Due to the dynamic nature of many applications executed on computational grids, it is not possible to know in advance the arrival date and the duration of each request. Therefore, in this context, it is neither possible to design a static algorithm that assumes the full knowledge of the application nor a dynamic one that only assumes the knowledge of task duration.

In this paper, we have proposed and studied a randomized resource broker for heterogeneous multilevel architectures where the arrival date and duration of the

requests are given by probabilistic distributions (with fixed mean). We extend the results and analysis from previous literature (mainly from [1]) by generalizing the field of application, studied so far. Our contribution is an extensive study of the behavior of the resource broker and the platform under such stochastic workload. More precisely, we have studied the saturated and nonsaturated case. For each case, we compute the average queue length of each computing element and study the number of CPUs used on each computing element. We analyze the behavior of the system when it switches from the saturated to the nonsaturated case. Finally, we evaluate how jobs are delayed according to the global load of the grid. Moreover, each of these metrics is studied both analytically and experimentally. To the best of our knowledge, these kinds of analysis and measurement have only been studied in homogeneous environments; our work extends these results to heterogeneous systems.

We have shown, by plotting together our simulation observations and our theoretical predictions or approximations, that we have acquired a really good knowledge of random job brokering.

Our future works are directed toward more complex cases: current-state dependent brokering (based on queue lengths, free CPUs, estimation of waiting time, etc.), (partially) randomized or fully deterministic, for other job length and interarrival distributions. These new constraints will more than likely make our analysis more difficult. Indeed, for instance, we will need to introduce feedback from computing elements to the resource broker. We also want to extend our work toward fault tolerance and reliability by duplicating requests on several computing elements or by restarting request when a failure occurs.

REFERENCES

[1] V. Berten and J. Goossens, "On the Job Distribution in Random Brokering for Computational Grids," *Proc. Second Int'l Symp. Parallel and Distributed Processing and Applications (ISPA 2004),* 2004.

[2] C. Ernemann, V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yah-yapour, "On Advantages of Grid Computing for Parallel Job Scheduling," *Proc. Second IEEE Int'l Symp. Cluster Computing and the Grid (CC-GRID 2002),* May 2002.

[3] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik, and P. Wong, "Theory and Practice in Parallel Job Scheduling," *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph, eds., pp. 1-34, Springer Verlag, 1997.

[4] W. Feller, *An Introduction to Probability Theory and Its Applications,* vol. 2, second ed. John Wiley & Sons, 1971.

[5] L. Gong, X.-H. Sun, and E.F. Watson, "Performance Modeling and Prediction of Nondedicated Network Computing," *IEEE Trans. Computers,* vol. 51, no. 9, pp. 1041-1055, Sept. 2003.

[6] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yah-yapour, "Evaluation of Job-Scheduling Strategies for Grid Computing," *Proc. First IEEE/ACM Int'l Workshop Grid Computing,* pp. 191-202, 2000.

[7] M. Harchol-Balter, "Task Assignment with Unknown Duration," *J. ACM,* vol. 49, no. 2, pp. 260-288, 2002.

[8] L. Hui, D. Groep, and L. Wolters, "Workload Characteristics of a Multi-Cluster Supercomputer," *Proc. 10th Workshop Job Scheduling Strategies for Parallel Processing, in conjunction with ACM Sigmetrics—Performance 2004,* June 2004.

[9] U.S.J. Krallmann and R. Yah-yapour, "On the Design and Evaluation of Job Scheduling Algorithms," *Job Scheduling Strategies for Parallel Processing,* pp. 17-42, 1999.

[10] H. Karatza, "Scheduling in Distributed Systems," *Proc. MASCOTS 2003,* pp. 336-356, 2004.

[11] L. Kleinrock, *Queuing Systems,* vols. 1-2, Wiley, 1975.

[12] A. Mu'alem and D. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 6, pp. 529-543, June 2001.

[13] R. Nelson, *Probability, Stochastic Processes, and Queueing Theory.* Springer-Verlag, 1995.

[14] N. Thomas, J. Bradley, and W. Knottenbelt, "Performance of a Semi Blind Service Scheduler," *Proc. UK e-Science All Hands Meeting,* 2004.

[15] N. Thomas, J. Bradley, and W. Knottenbelt, "Stochastic Analysis of Scheduling Strategies in a Grid-Based Resource Model," *IEE Proc. Software,* 2004.

Vandy Berten received the masters degree in computer science from the Université Libre de Bruxelles, Belgium (ULB), in September 2003. He started his PhD degree in October 2003 under the supervision of Joël Goossens. His main research focuses on grid computing and grid brokering (or scheduling). He has been a research fellow (aspirant) of the Belgian National Fund for Scientific Research (FNRS) since October 2003 at the ULB.

Joël Goossens received the MSc degree in computer science in 1992, the MSc degree in network and management in 1993, and the PhD degree in computer science in 1999, all from the Université Libre de Bruxelles, Belgium. He has been an assistant professor at the Université Libre de Bruxelles since October 2001. He teaches algorithms and programming, database, compiler design, and real-time scheduling. His main research interests are presently in real-time scheduling theory, computer integrated manufacturing, memory management (garbage collection), and scheduling for grids.

Emmanuel Jeannot received the PhD and masters degrees of computer science (respectively, in 1996 and 1999), both from Ecole Normale Supérieure de Lyon and both under the advisory of Michel Cosnard. He has been an associate professor at the Université Henry Poincaré, Nancy 1, since September 2000. He is doing research at the INRIA-CNRS LORIA laboratory. His main research interests are scheduling for heterogeneous environments and grids, data redistribution, grid computing softwares, adaptive online compression, and programming models.

Messages Scheduling for Parallel Data Redistribution between Clusters

Johanne Cohen, Emmanuel Jeannot, Nicolas Padoy and Frédéric Wagner

***Abstract*— We study the problem of redistributing data between clusters interconnected by a backbone. We suppose that at most k communications can be performed at the same time (the value of k depending on the characteristics of the platform). Given a set of messages we aim at minimizing the total communication time assuming that communications can be preempted and that preemption comes with an extra cost. Our problem, called *k-Preemptive Bipartite Scheduling (KPBS)* is proven to be NP-hard. We study its lower bound. We propose two $\frac{8}{3}$-approximation algorithms with low complexity and fast heuristics. Simulation results show that both algorithms perform very well compared to the optimal solution and to the heuristics. Experimental results, based on an MPI implementation of these algorithms, show that both algorithms outperform a brute-force TCP based solution, where no scheduling of the messages is performed.**

***Index Terms*— Message scheduling ; data redistribution ; grid computing ; approximation algorithm ; code coupling.**

I. Introduction

In recent years, the emergence of cluster computing, coupled with fast wide area networks has allowed the apparition of grid computing, enabling parallel algorithms to take advantage of various distant resources, be it computing power, software or data. However the classical problem of minimizing the communications/computations ratio remains and is even more difficult since communication times increase on slower networks. It is therefore important to try to minimize communication times. In this work we focus on the scheduling of the messages when a parallel data redistribution has to be realized on a network, called a backbone. Two parallel machines are involved in the redistribution: the one that holds the data and the one that will receive the data. If the parallel redistribution pattern involves a lot of data transfers, the backbone can become a bottleneck. Thus, in order to minimize the parallel data redistribution time and to avoid the overloading of the backbone it is required to schedule each data transfer.

The message scheduling problem appears in the context of data redistribution but also in the context of packet switching for wavelength-division multiplexed (WDM) optical network [7], [13], [23], [26], [28] or for satellite-switched time division multiple access (SS/TDMA) [4], [15], [16]. The solution proposed in this paper works for these two cases.

Data redistribution has mainly been studied in the framework of high performance parallel computing [1], [8], [10]. In this paper we study a generalization of the parallel data redistribution. Indeed, contrary to some previous works that only deal with block-cyclic redistribution [3], [10], [24], here no assumption is made on the redistribution pattern. Moreover, some workk [1], [8] assume that there is no bottleneck. In this paper, it is not the case and we suppose that the ratio between the throughput of the backbone and the throughput of each of the n nodes of the parallel machines is k. Hence, at most k communications can occur at the same time. We study the problem for all values of k. We focus on the case $k < n$ (the backbone is a bottleneck) whereas the case $k \geq n$ has been tackled in [1], [8].

Redistributing data between clusters has recently received considerable attention as it occurs in many application frameworks. We provide here three examples of such frameworks taken from distributed code-coupling, parallel task execution and persistence and redistribution for metacomputing:

1) Distributed code coupling: Code coupling applications [21] are composed of several codes that interact with each other. They are used for simulating complex systems. Such a system is composed of several models, each model being simulated by one parallel code or component [30]. Moreover, in distributed code coupling, each code/component is running on a different parallel machine or cluster [2], [25]. For instance the hydrogrid project [20] aims at modeling and simulating fluid and solute transport in subsurface geological media. In this application two parallel codes are coupled: one for flow simulation and one for transport simulation. For performance reasons, each parallel code requires a very large cluster to execute on: they cannot execute on the same cluster. During the simulation the models interact with each-other and therefore the parallel codes need to exchange data. Hence, this exchange of data is a redistribution between distant clusters.
2) Parallel task execution: Recent works in the field of mixed parallelism [31], [27], [5], have shown the potential of executing data parallel tasks concurrently on different clusters. In some cases, these tasks need to communicate with each other and data has to be redistributed between each cluster that host the task.
3) Persistence and redistribution in GridRPC systems: Several metacomputing environments implement the client-agent-server model [6], such as Netsolve or Ninf [18]. In this model the agent has to map a client request to a server, the request is then being processed in an RPC way. One of the drawback of this approach is that data are sent back to the client at the end of every computation. This implies unnecessary communications

This work is partially funded by the INRIA ARC redGRID, the ACI GRID and the Rgion Lorraine

when computed data are needed by an other server in further computations. Some recent enhancements of this model deal with data management and allow data to be persistent on the server [9], [11]. As the service can be parallel the data can be distributed among the node of the cluster and when this data is required for further computation on a distant server. In this case a parallel data redistribution occurs between distant clusters.

The contribution of this paper is the following. We prove that the problem of scheduling any parallel data redistribution pattern is NP-hard for any value of $k(k < n)$. We exhibit a lower bound for the number of steps of the redistribution and a lower bound for the sum of the durations of each step. Next, we propose two algorithms (called GGP and OGGP) that have a $\frac{8}{3}$-approximation bound. On the other hand we study simple and fast heuristics that achieve a good average performance. Simulation results show that both GGP and OGGP outperform the heuristics and are close to the optimal solution. Moreover, we have implemented these algorithms and tested them on real examples using MPI. Results show that we outperform a TCP-based brute-force solution that consists in letting the transport layer doing the scheduling and managing alone the congestion.

II. Description of the Problem

A. Modelization of the Problem

We consider the following heterogeneous architecture made of two clusters of workstations $\mathcal{C}_1$ and $\mathcal{C}_2$ connected together by a backbone of throughput D. Let n_1 be the number of nodes of $\mathcal{C}_1$ and n_2 be the number of nodes of $\mathcal{C}_2$ (and $n = n_1 + n_2$). All the nodes of the first cluster have a throughput d_1 and the nodes of the second have a throughput d_2.

We assume that any node of $\mathcal{C}_1$ can communicate to any node of $\mathcal{C}_2$. This requires that each node has its own address (as in GRID'5000[1] or for icluster project[2]) or, if the nodes are behind a NAT, that the router/front-end implements a specific port forwarding mechanism to each node. The bottleneck that might come from such mechanism can easily be captured by our model.

Let us consider a parallel application that must execute the first part of its computations on $\mathcal{C}_1$ and the second part on $\mathcal{C}_2$. This is the case where an application is made of several parallel components, data parallel tasks or requests with dependencies. During the execution of the application parallel data must be redistributed from the first cluster to the second one.

We assume that the communication pattern of the redistribution is computed by the application. We focus on efficiently transmitting the data, not on computing the pattern itself. For computing the pattern in the case of block-cyclic redistribution see [17]. This pattern is modeled by a *traffic matrix* $T = (t_{i,j})_{1 \leq i \leq n_1, 1 \leq j \leq n_2}$, where $t_{i,j}$ represents the amount of information that must be exchanged between node i of cluster $\mathcal{C}_1$ and node j of cluster $\mathcal{C}_2$.

[1] www.grid5000.org
[2] icluster.imag.fr

For a given traffic pattern and for a particular architecture our goal is to minimize the total transmission time. In order to perform the redistribution, one naive solution consists in sending all the data from all the nodes of $\mathcal{C}_1$ to all the nodes of $\mathcal{C}_2$ at the same time and let the transport layer (for instance TCP) schedule the segments. This solution, as we will show in the results section, is suboptimal for many reasons. If the traffic matrix is very large, dense with high coefficient a lot of traffic is generated at the same time. This traffic cannot be handled either by the backbone (when the aggregated bandwidth of the emitting card is greater than the bandwidth of the backbone) or by the cards themselves (when the incoming traffic has a throughput greater than the throughput of a given card). In both cases, TCP segments will be dropped. TCP will detect the problem and starts to control the congestion by reducing the window size and therefore reduce the amount of data sent at a given time. To avoid these problems, we use the knowledge we have (i.e. the traffic matrix) to perform optimizations at the application level and control by ourselves the congestion by defining a schedule for all the communications.

We consider two constraints relative to the communications:

1) **the 1-port constraint**. A transmitter (resp. a receiver) cannot perform more than one communication at a given moment. However, more than one communication can occur at the same time as long as the receiver/transmitter pair is different. A parallel transmission of messages between different pairs is called a *step*.
2) **the k constraint**. The maximum number of communications that can occur during a step is denoted by k. This number depends mainly on the ratio D/d_1 and D/d_2. It comes from the fact that no congestion occurs when the aggregated bandwidth generated by cluster $\mathcal{C}_1$ or received by $\mathcal{C}_2$ is not larger than the bandwidth D of the backbone. Therefore, k must respect the following equations: (a) $kd_1 \leq D$, (b) $kd_2 \leq D$, (c) $k \leq n_1$ and (d) $k \leq n_2$.

We denote by d the speed of each communication $d = \min(d_1, d_2, D)$. For instance let us assume that $n_1 = 200$, $n_2 = 100$, $d_1 = 10$Mbit/s, $d_2 = 100$Mbit/s and $D = 1$GBbit/s ($D = 1000$Mbit/s). In that case, $k = 100$ because $\mathcal{C}_1$ can send 100 outgoing communications at 10 Mbit/s generating a total of 1 Gbit/s aggregated bandwidth and each network card of $\mathcal{C}_2$ can receive the data at $d = 10$ Mbit/s.

A common approach to minimize the overall transmission time is to enable preemption, i.e. the possibility to interrupt the transmission of a message and to complete it later. In practice, this involves a non-negligible cost, called *startup delay* and denoted here by β, which is the time necessary to start a new *step*.

B. Formulation of the Problem

Let T be a traffic matrix, k be the maximum number of communications at each step, β be the startup delay and d be the speed of each communication.

We can normalize the problem by d: The traffic matrix T, can be replaced by the matrix $\mathcal{M} = (m_{i,j}) = (\frac{t_{i,j}}{d})_{1 \leq i \leq n_1, 1 \leq j \leq n_2}$ that represents the communication time.

Before describing the problem formally, we need to introduce some terminology on graphs. Let $G = (V_1, V_2, E)$ be a bipartite graph with vertex set $V_1 \cup V_2$ and edges set $E \subseteq V_1 \times V_2$: if $(i, j) \in E$ then $i \in V_1$ and $j \in V_2$. A set $M \subseteq E$ is called a *matching* if no vertex $v \in V_1 \cup V_2$ is incident with more than one edge in M. A matching is called *perfect* if all vertices of $V_1 \cup V_2$ are incident to exactly one edge in M. A *weighted matching* (M, w) is a matching associated to a function that gives the weight of all its edges: $w : M \to \mathbb{Q}^+$.

The matrix $\mathcal{M}$ is equivalent to a *weighted bipartite graph* $G = (V_1, V_2, E, w)$ (see Fig. 1) where $w : E \to \mathbb{Q}^+$, $\forall e = (i, j) \in E$, $w(e) = m_{i,j}$. Each node of cluster $\mathcal{C}_1$ (resp. $\mathcal{C}_2$) is represented by a node of V_1 (resp. V_2). Hence, $|V_1| = n_1$ and $|V_2| = n_2$.

	1'	2'	3'	4'
1	2	0	0	0
2	0	1	1	0
3	0	1	0	0
3	0	0	0	1

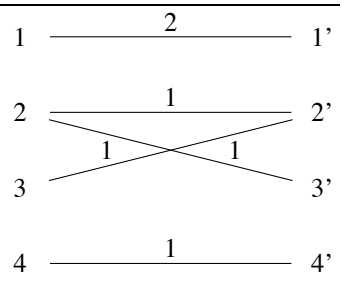


Fig. 1. *Correspondance between matrix $\mathcal{M}$ and a bipartite graph.*

Given a weighted bipartite graph $G = (V_1, V_2, E, w)$ that represents the communication to execute with their time, we tackle the problem of scheduling these communications. The solution must describe when a node of V_1 must communicate to a node of V_2 and for how long. The solution will be composed of steps. Each step needs to follow the constraints presented in the previous section:

- The 1-port model avoids communication contention: during one step a given node cannot communicate with more than one node. Since an edge of graph G represents an communication between two nodes, a step will be modeled by a matching of G.
- Moreover, the backbone is a bottleneck: at most k communications can occur during one step. Hence, a communication step will be represented by a matching with at most k edges.

We recall that the preemption is allowed: a communication between two given nodes might be stopped and resumed later. Therefore, a given edge of G might occur in several matchings, each representing a different communication step. In order to describe which part of the whole communication is done during a given step we add weight function to each the matching. These weights refer to the amount of exchanged data. Hence, an edge of G can be decomposed and be present into several matchings provided that the sum of all the weights of this edge in these matchings is not smaller than the weight of the original edge in G.

We denote the matching and its weignted function corresponding to a communication step by a *valid weighted matching* (for the remaining, a valid weighted matching contains at most k edges).

In order to execute a schedule we execute each step one after the other. Step i corresponds to a valid weighted matching (M_i, w_i) and for each edge $e = (u, v) \in M_i$ we send data between node u of cluster $\mathcal{C}_1$ to node v of cluster $\mathcal{C}_2$ for a time equal to $w_i(e)$ (or $w_i(e) * d$ amount of data). We execute step $i + 1$ when all communications of step i are done. The duration of step i is therefore $\beta + W_i$ where β is the startup cost of a communication and $W_i = \max_{e \in M_i} w_i(e)$ Fig. 2 gives an example of a valid schedule.

We call this optimization problem *k-Preemptive Bipartite Scheduling (KPBS)*, formally defined as follows:

Given a weighted bipartite graph $G = (V_1, V_2, E, w)$ where $w : E \to \mathbb{Q}^+$, an integer[3] $k \geq 2$ and a rational β, find a collection $\{(M_1, w_1), (M_2, w_2), \ldots, (M_s, w_s)\}$ of valid weighted matchings such that:

1) The matchings are a decomposition of E: $\cup_{i=1}^{s} M_i = E$
2) all the functions w_i, $1 \leq i \leq s$, must respect the following inequality: $\forall e \in E$, $\sum_{i \in \{j | e \in M_j\}} w_i(e) \geq w(e)$.
3) Any matching M_i contains at most k edges ($|M_i| \leq k$, $i \in [1, s]$) and its cost is equal to the rational number $\beta + W_i$, where $W_i = \max_{e \in M_i} w_i(e)$.
4) $(\sum_{i=1}^{s} (\beta + W_i))$ is minimized.

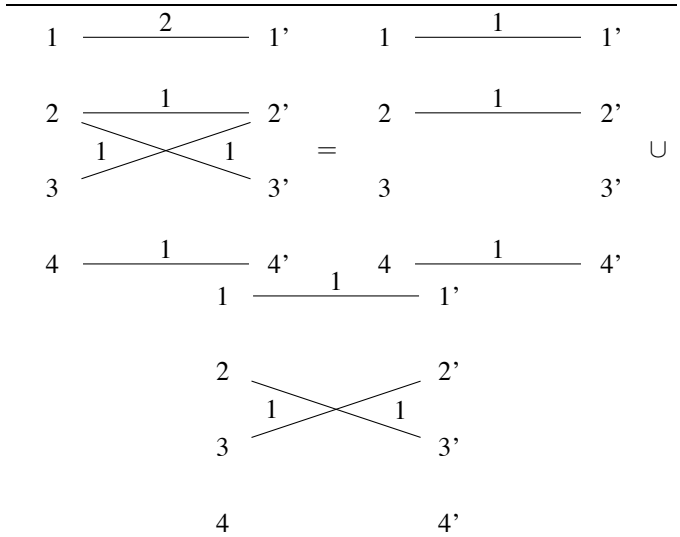


Fig. 2. *An example for KPBS problem with the graph of Fig. 1 and $k = 3$. The solution contains two steps and the useful transmission cost is equals to 2 $(= 1 + 1)$. The cost of the solution is $2 + 2\beta$. Thanks to preemption, the edge of cost 2 is decomposed into two steps.*

a) Case $\beta = 1$: It is important to see that solving the case $\beta = 1$ is sufficient to consider. Indeed, if $\beta \neq 1$, one can divide (normalize) all the edges weights by β, then solve the problem assuming $\beta = 1$ and multiply the edges weight of the matchings of the solution by the original value of β. Therefore, in the following of this paper we will consider that $\beta = 1$. β will not appears in the NP-completeness proof or as a parameters of the algorithms (Sections IV and VI).

[3]the case $k = 1$ is not interesting because the backbone is saturated by one communication

In the remainder of this paper, we use the following notation: for any solution S of $KPBS$, the cost of S is $\alpha + s$ (β is considered equals to 1), where s is the *number of steps* and α is the *useful transmission cost*.

III. Related Work

This problem has been partially studied in the context of Satellite-Switched Time-Division Multiple access systems (SS/TDMA) [4], [15], [16]. In [4], the problem with $\beta = 0$ is studied and an optimal algorithm with $O(mn)$ steps is described. In [15] an optimal algorithm that finds the minimal number of steps is described. In [16] the problem without preemption is studied. It is shown NP-hard and a heuristic is given.

The KPBS problem partially falls in a field originated by packet switching in communication systems for optical network called wavelength-division multiplexed (WDM) broadcast network [7], [13], [23], [26], [28]. The problem of minimizing the number of steps is studied in [13], [15], and the problem of minimizing the total cost is studied in [23]. In [7] and in [26], the authors consider a version of the KPBS problem where the number of receivers is equal to the number of messages that can be transmitted at the same time ($k = n_2$) and where the setup delay can be overlapped by the communication time (In [26] authors also assume that all messages have the same size). In that case, a list-scheduling algorithm is proven to be a 2-approximation algorithm [7]. The case where the backbone is not a constraint ($k \geq \min(n_1, n_2)$) has been studied in [1], [8] and it is known as the *preemptive bipartite scheduling (PBS)*. PBS was proven to be NP-hard in [12], [16]. Approximating the PBS problem within a ratio number smaller than $\frac{7}{6}$ has been proven impossible unless $P =$ NP [8]. Several approximation algorithms for the PBS problem have been proposed in the literature. In [8], two different polynomial time 2-approximation algorithms for PBS have been proposed and in [1], an improvement of this result is given.

In [19] the problem of mapping the data to the processors for minimizing the communications is studied in the context of local block cyclic redistributions. It aims at minimizing the amount of data to transfer and not the communication time.

In the context of block cyclic redistribution many works exist (see [3], [10], [24] for example). In this case the backbone is not a constraint and the redistribution pattern is not arbitrary. Hence, all these problems are less general than KPBS.

IV. Complexity Results

This problem has already been proven NP-hard for the particular case where $k \geq \min(n_1, n_2)$ [12], [16]. We prove that it remains NP-hard for any fixed $k \geq 2$ (with a different reduction than in [12], [16]). The *decision* problem of KPBS (D-KPBS) is defined as follow:

Instance: A weighted bipartite graph $G = (V_1, V_2, E, w)$ where $w : E \to \mathbb{Q}^+$, an integer k, a rational number B.
Question: Is there a collection

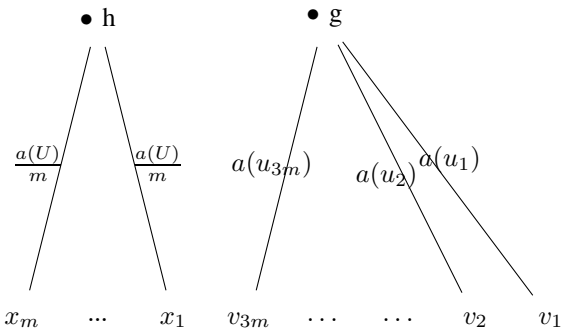


Fig. 3. An example of graph G from an instance (U, s) of Partition Problem.

$\{(M_1, w_1), (M_2, w_2), \ldots, (M_s, w_s)\}$ of valid weighted matchings such that $E = \cup_{i=1}^{s} M_i$ and $\sum_{i=1}^{s} W_i + s \leq B$. with for any $e \in E$, $\sum_{i \in \{j | e \in M_j\}} w_i(e) \geq w(e)$?

Theorem 1: Let $k \geq 2$ be a fixed integer. D-KPBS is NP-complete in the strong sense.

Proof of Theorem 1:: It is easy to see that D-KPBS belongs to NP. We show that it can be reduced to the *3-Partition* problem [14] defined as follow:

Instance: A finite set $U = \{u_1, u_2, \ldots, u_{3m}\}$ and a size $a(u) \in \mathbb{Z}^+$ for each $u \in U$.
Question: Can U be partitioned into m disjoint sets $U_1, \ldots, U_m$ such that: For $1 \leq i \leq m$, $\sum_{u \in U_i} a(u) = \frac{1}{m} \sum_{u \in U} a(u)$?

Let $a(U) = \sum_{i=1}^{3m} a(u_i)$. We transform 3-partition to D-KPBS: Let $U = \{u_1, u_2, \ldots, u_{3m}\}$ be a finite set and a size $a(u) \in Z^+$ for each $u \in U$ in an arbitrary instance of *3-Partition* problem. Now, an instance of D-KPBS are constructed from set U and function a. We set $B = a(U) + 3m$ and $k = 2$. We consider the following weighted bipartite graph $G = (V_1, V_2, E, w)$:

- $V_1 = \{v_1, v_2, \ldots, v_{3m}, x_1, \ldots, x_m\}$ and $V_2 = \{g, h\}$
- $E = \{(x_i, h) : 1 \leq i \leq m\} \cup \{(v_i, g) : 1 \leq i \leq 3m\}$
- $w(x_i, h) = a(U)/m$ for $1 \leq i \leq m$
- $w(v_i, g) = a(u_i)$ for $1 \leq i \leq 3m$

Fig. 3 gives an example of this transformation. G can clearly be constructed in polynomial time. We claim that the instance of D-KPBS admits a solution if and only if the instance of *3-Partition* has a desired partition.

($\Rightarrow$) Let $\{U_1, \ldots, U_m\}$ be a solution of the *3-Partition* instance. Then a collection $\{(M_i, w_i) : 1 \leq i \leq 3m\}$ of valid weighted matchings is defined as follows: for $1 \leq i \leq 3m$, $M_i = \{(v_i, g)(x_j, h)\}$, $w_i((v_i, g)) = a(u_i)$, $w_i((x_j, g)) = a(u_i)$, where $u_i \in U_j$. This collection is a solution of D-KPBS. Indeed the sum of these matchings is $a(U)$ because $\sum_{j=1}^{m} \sum_{u_i \in U_j} a(u_i) = a(U)$. The cost of this solution is exactly B.

($\Leftarrow$) Conversely, we suppose that the instance of D-KPBS admits a solution. Then the useful transmission cost is at least equal to $a(U)$ because of the edges incident to vertex h. There are at least $3m$ steps, because vertex g has $3m$ neighbors.

Since the cost is lower than or equal to B, both previous inequalities are equalities. Therefore, for $1 \leq i \leq 3m$, no edge incident to v_i can be split and the solution of the instance of D-KPBS is composed of $3m$ valid matchings. Thus the solution having the desired properties can be written $(M_i)_{1 \leq i \leq 3m}$. Now, we will determine weight functions $(w_i)_{1 \leq i \leq 3m}$. Since the size of a matching is at most $2(= k)$, for $1 \leq i \leq 3m$, all valid matchings C_i of the solution contains only one edge incident to g and to u_j (w.l.g. we assume that $j = i$), and thus $w_i((v_j, g)) = a(u_i)$. Necessarily M_i contains an edge incident to one vertex belonging to $\{x_1, \ldots, x_m\}$, having the weight $a(u_i)$ (the same weight as the other edge of the matching).

For $1 \leq j \leq m$, let U_j be the set of the u_i such that M_i contains an edge adjacent to x_j. Then, for $1 \leq j \leq m$, we have $\sum_{u_i \in U_j} a(u_i) = \sum_{u_i \in U_j} w_i((v_i, g)) = \frac{a(U)}{m}$. Hence, sets $U_1, \ldots, U_m$ (such that for $1 \leq j \leq m$, $\sum_{u \in U_j} a(u) = \frac{1}{m} a(u)$) is a partition of U .

Since for any k fixed, we have the same proof, Theorem 1 is proven. ∎

Since the problem decision problem D-KPBS is NP-complete, the optimization problem KPBS is a NP-hard. The remainder of this paper is devoted to find approximation algorithms and to experiment them.

V. Lower Bounds

Before giving a lower bound for the optimal solution, we give some graph notations. We define the weight $w(v)$ of a node v of G to be the sum of weights of all edges incident to vertex v. We denote the maximum of $w(v)$ over all vertices by $W(G)$. Let $P(G)$ be the sum of the weights of all edges of graph G. We denote the maximum degree of the bipartite graph G by $\Delta(G)$, its number of edges by $m(G)$ and its number of vertices by $n(G)$. For example, in Fig. 1, $W(G) = 2$, $P(G) = 6$ and $\Delta(G) = 2$.

Proposition 1: Let $G = (V_1, V_2, E, w)$ be a weighted bipartite graph. Let k be an integer, β a rational. The cost of the optimal solution for the instance $\langle G, k, \beta \rangle$ of KPBS is at least $\eta(G) = \eta_d(G) + \beta\eta_s(G)$ with

$$\eta_d(G) = max\left(W(G), \frac{P(G)}{k}\right)$$

$$\eta_s(G) = max\left(\Delta(G), \left\lceil \frac{m(G)}{k} \right\rceil\right)$$

Proof of Proposition 1: $\eta_s(G)$ is a lower bound for the number of steps. The first term of the maximum accounts for the fact that two edges incident to the same node cannot appear in the same step and the second term for the fact that a step contains at most k edges. $\eta_d(G)$ is a lower bound for the useful transmission cost and is obtained similarly. The total cost is therefore minimized by $\eta_d(G) + \beta\eta_s(G)$. ∎

VI. Algorithms

In this section we present two algorithms we propose to use for the KPBS problem. As discussed in before, we consider only the case $\beta = 1$ here. We start by presenting the *GGP* (Generic Graph Peeling) which is a polynomial time $\frac{8}{3}$-approximation algorithm. This algorithm is relatively complex hence we describe it relying on a subalgorithm called *weight-regular extension algorithm* in order to simplify the presentation.

We first introduce the main ideas behind these algorithms together with a more formal description of the different steps.

We then continue this section by an analysis of the different properties of GGP. Three key properties are studied: algorithm correctness. approximation ratio, and worst case complexity. In order to study the approximation ratio we need to introduce another algorithm called *multigraph algorithm*. We prove that this algorithm is a pseudo polynomial $\frac{8}{3}$-approximation and that GGP can always give better results than it.

Finally we introduce the *OGGP* algorithm (Optimized GGP) which is a direct enhancement of GGP and compute its worst case complexity.

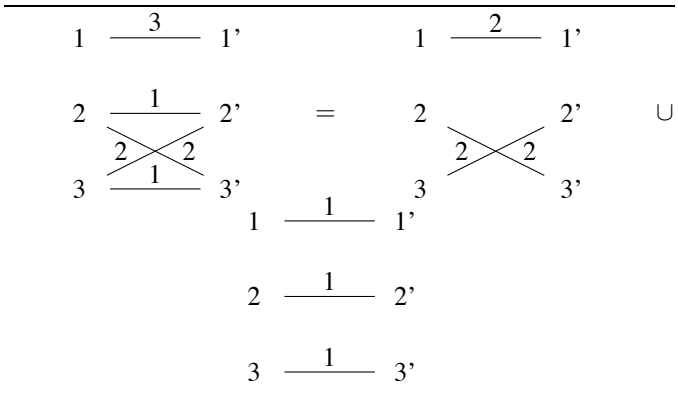


Fig. 4. *Peeling a weight-regular graph leads to a weight-regular graph*

A. GGP Algorithm

1) Simple Case where G is weight-regular, with all edges of integer weights: Solving the KPBS problem is easy in the case where there is no constraint on k (i.e. $k = n$) and the input graph G holds the following properties: G is weight-regular, with all edges of integer weights. A weight-regular graph is a graph such that for each of its nodes the sum of all weights of adjacent edges is the same (see Fig 4).

The algorithm is based on an interesting propriety: any weight-regular graph has a perfect matching [8]. We are therefore able to pick such a perfect matching which can be communication step. But, as all communications in a given step should to be ended simultaneously (to minimize waiting and therefore overall cost) we cut the duration of all communications (i.e. the weight of any edge in the matching) to the smallest one. By doing this, the graph left after removing the matching is still weight-regular becausewe removed the same amount of weight on each node (see Fig. 4). We then start again, removing another matching from the graph. In the remaining of this paper, we call *peeling* a graph this procedure of step by step removing perfect matchings from it. The algorithm ends when the graph is empty. In Fig 4, we removed a perfect matching of weight 2 for all edges leading to a weight-regular graph (which is also a perfect matching).

Input: A bipartite graph $G = (V_1, V_2, E, w_G)$, an integer k.
Output: A set of valid weighted matchings S.

let $V_1 = \{v_1, \ldots, v_{n_1}\}$, $V_2 = \{u_1, \ldots, u_{n_2}\}$, where $n_1 = |V_1|$, $n_2 = |V_2|$

1. Build a graph $H = (V_1, V_2, E, w_H)$ such that $\forall e \in E, w_H(e) = \lceil w_G(e) \rceil$

2. Build a graph $I = (V_{1_I}, V_{2_I}, E_I, w_I)$ with $\frac{P(I)}{k} \geq W(I)$ and $\frac{P(I)}{k} \in \mathbb{N}$:

$\phi = \max\left(W(H), \left\lceil \frac{P(H)}{k} \right\rceil\right)$

δ: number of nodes added to V_1 and V_2: $\delta = \left\lceil \frac{\phi.k - P(H)}{W(H)} \right\rceil$

$V_{1_I} = \{v'_1, \ldots, v'_{n_1+\delta}\}$, $V_{2_I} = \{u'_1, \ldots, u'_{n_2+\delta}\}$

$E_I = E_1 \cup E_2$, where

$E_1 = \left\{(v'_i, u'_j) | (v_i, u_j) \in E, i \in [1, n_1],\ j \in [1, n_2]\right\},\ \forall (v'_i, u'_j) \in E_1, w_I\left((v'_i, u'_j)\right) = w\left((v_i, u_j)\right)$

if $\delta = 0$ then

$E_2 = \emptyset$

else

$E_2 = \left\{(v'_{n_1+i}, u'_{n_2+i}) | i \in [1, \delta]\right\}$

if $\delta \neq 1$ then

$\forall i \in [1, \delta - 1], w_I\left((v'_{n_1+i}, u'_{n_2+i})\right) = W(H)$

if $(\phi.k - P(H)) \bmod W(H) \neq 0$ then

$w_I\left((v'_{n_1+\delta}, u'_{n_2+\delta})\right) = (\phi.k - P(H)) \bmod W(H)$

else

$w_I\left((v'_{n_1+\delta}, u'_{n_2+\delta})\right) = W(H)$

3. Transform I into a $\frac{P(I)}{k}$-weight-regular graph $J = (V_{1_J}, V_{2_J}, E_J, w_J)$ using the algorithm described in Fig 6.

4. $S = \emptyset$

5. While $E_J \neq \emptyset$ do:

5.1. Choose a perfect matching M in J

5.2. Change w_M such that $\forall e \in M, w_M(e) = s(M)$ the smallest weight of the edges of M

5.3. Add M to S, the set of solution matchings

5.4. $\forall e \in M$ change $w_J(e)$ to $w_J(e) - w_M(e)$

5.5. Remove from E_J all edges of weight 0

6. Remove all edges e in S such that $e \notin E$

Fig. 5. **GGP algorithm**

2) General Case: For the general case, we start by modifying the input graph to obtain a weight-regular graph as in the simple case. We do so by taking into account latency and the k-constraint.

The difficulty of the KPBS problem comes from the startup delay cost associated to each step. This means that in order to be efficient we should avoid generating a too high number of steps and therefore avoid cutting any edge into too little pieces. In particular

we do not want to subdivise an edge with a cost already lower than the startup delay cost (which has a value of 1 due to normalization). To achieve that, the first step of the GGP algorithm is to round all weights on all edges to their next upper integers and after that considering only matchings with integer costs in the algorithm.

The other main objective of the GGP algorithm is to avoid having more than k edges in a matching. In order to do that, we add some *virtual* edges in the input graph. By choosing them carefully we can ensure that any perfect matching will contain at most k *real* edges (i.e. edges from the original graph): see section VI-B.1.

Input: The weighted bipartite graph $I = (V_{1_I}, V_{2_I}, E_I, w_I)$ of step 2 of GGP, an integer k.
Output: A weighted bipartite graph J such that J is a $\frac{P(I)}{k}$ weight-regular bipartite graph.

1. Copy the graph I in J: $V_{1_I} = V_{1_J}, V_{2_I} = V_{2_J}$, $E_I = E_J$, and $\forall e \in E_J, w_J(e) = w_I(e)$.
2. We use a variable cn which is a node. cn starts being undefined.
3. Now foreach $s \in V_{1_I}$(s also in V_{1_J}) do:
   3.1 Compute $mw(s)$.
   If cn is undefined or $mw(cn) = 0$ then
      3.1.1 add a new node n to V_{2_J}
      3.1.2 $cn = n$
      3.1.3 add an edge (s, cn) to E_J with $w_J(s, cn) = mw(s)$
   else
      3.1.4 if $mw(cn) \geq mw(s)$ then
         3.1.4.1 add an edge (s, cn) to E_J with $w_J(s, cn) = mw(s)$
      else
         3.1.4.2 add an edge (s, cn) to E_J with $w_J(s, cn) = mw(cn)$
         3.1.4.3 add a new node n to V_{2_J}
         3.1.4.4 $cn = n$
         3.1.4.5 add an edge (s, cn) to E_J with $w_J(s, cn) = mw(s)$
4. Do the same for all nodes in V_{2_J}.

Fig. 6. **Weight-regular extension algorithm**

Hence if we put all that into order, GGP is divided into these 4 large steps:

1) Building graph H by rounding all weights (step 1 in the formal description of Fig 5),
2) Building graph I for preparing step 3 (step 2 in the formal description),
3) Build a weight-regular graph J while also adding virtual edges taking care of the k constraint (step 3 in the formal description, and Fig 6),
4) Peel the graph (steps 4, 5 and 6 in the formal description).

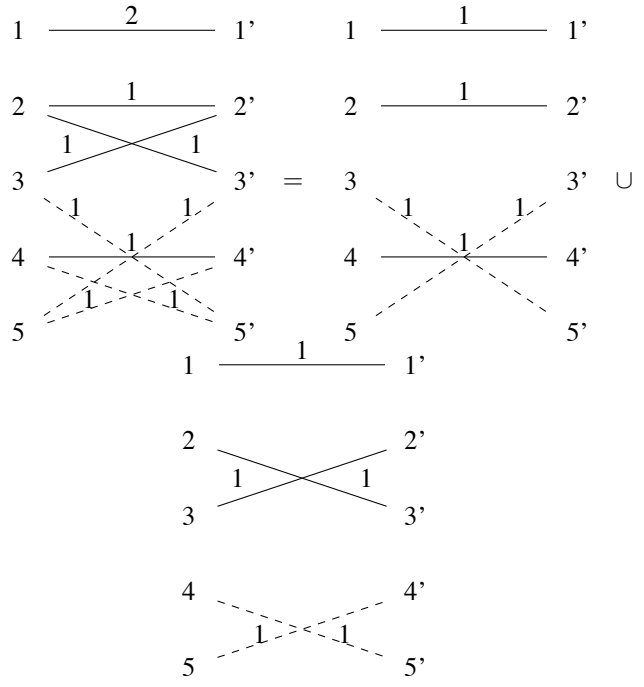


Fig. 7. *Example of $k = 3$ constraint solving. The Graph J (after step 3 of GGP) is built from the graph of Fig. 1 and peeled to obtain the solution (unbroken edges) of Fig. 2.*

To build the weight-regular graph of step 3 we need to define the function $mw : V_1 \cup V_2 \rightarrow \mathbb{N}$ the function assigning to a node s the *missing weight* $mw(s)$ of this node for the graph to be $\frac{P(I)}{k}$ weight-regular. We have $mw(s) = \frac{P(I)}{k} - w(s)$. The algorithm building the weight-regular graph is shown in Fig 6.

a) Example: Consider the example of Fig. 7. The input of GGP is the graph given on Fig. 1 with $k = 3$. After step 3 of GGP, nodes 5 and $5'$ have been as well as edges shown as dashed lines have been added to the original graph. Then, we start peeling the graph and in any perfect matching the number of real edges is always $k = 3$ and the number of virtual edges is 2. The final solution (real edges) is the same as the one given Fig. 2

B. Properties

We start by proving that the weight-regular extension algorithm of Fig. 6 is correct.

Proposition 2: J is $\frac{P(I)}{k}$-weight-regular.

Proof of Proposition 2: We need to prove $\forall s \in V_{1_J} \cup V_{2_J}, w(s) = \frac{P(I)}{k}$. We consider two cases: the case where s was already in I and the case where s is a new node.

If s was already in I, the algorithm implies $mw(s) = 0$ and therefore $w(s) = \frac{P(I)}{k}$.

If s was not in I: the weights on the added edges is the sum of the missing weights for all nodes of one side, that is $\sum_{t \in V_{1_I}} mw(t)$ and $\sum_{t \in V_{2_I}} mw(t)$. We know that $\sum_{t \in V_{1_I}} mw(t) = \sum_{t \in V1_I} \frac{P(I)}{k} - w(t)$. It is therefore equal to $|V_{1_I}|\frac{P(I)}{k} - P(I) = (|V_{1_I}| - k)\frac{P(I)}{k}$ since there is no edge between two nodes on the same side. This value divided by $\frac{P(I)}{k}$ gives $(|V_{1_I}| - k)$ which means it is possible to add edges to $(|V_{1_I}| - k)$ new nodes s with $w(s) = \frac{P(I)}{k}$. The same reasoning holds for V_{2_I}. As we build all nodes sequentially, never leaving a node s with $w(s) < \frac{P(I)}{k}$ we have $\forall s \in V_{1_J} \cup V_{2_J}, w(s) = \frac{P(I)}{k}$. ∎

1) Correctness of GGP: The correctness of GGP follows from the respect of the 1-port and the k-constraint. The 1-port constraint is ensured by choosing matchings. The respect of the k-constraint requires the following lemma.

Lemma 1: Any perfect matching on J contains k edges belonging to I.

Proof of Lemma 1: Let M be a perfect matching on J. We have from proof of Proposition 2 that V_{1_J} is V_{1_I} with $|V_{2_I}| - k$ new nodes. Similarly V_{2_J} is V_{2_I} with $|V_{1_I}| - k$ new nodes. Therefore we have $|V_{1_J}| = |V_{2_J}| = |V_{1_I}| + |V_{2_I}| - k$ which is the number of edges in M. We know that none of the nodes added are connected together and also that any edge connected to a new node is not in E_I. Therefore we have one edge for each node added that is in M and not in E_I. Since we added $|V_{1_I}| - k$ and $|V_{2_I}| - k$ nodes the number of nodes in M and in E_I is $|V_{1_I}| + |V_{2_I}| - k - (|V_{1_I}| - k) - (|V_{2_I}| - k) = k$. ∎

Since I is built by adding edges from H, M has at most k edges belonging to G. To build the solution matchings on J from which all edges not belonging to G are removed. Therefore, any matching of the solution given by GGP respects the k-constraint.

2) Approximation Ratio: We define the $trans$ function which takes a weighted bipartite graph $G = (V_1, V_2, E, w_G)$ as argument and returns the corresponding multigraph $G' = (V_1', V_2', E')$ by splitting all edges such that an edge $e \in E$ of weight $w_G(e)$ is turned into $w_G(e)$ edges of weight 1.

With this function we can now define the Multigraph algorithm (Fig. 8). Basically this algorithm starts by constructing the same graph J as GGP but is different in the ways it peels the graph. We build $J' = trans(J)$ which is a regular graph and peel it into perfect matchings using the property that there always exist a perfect matching on a regular graph. Thus, we obtain a set of matchings whose costs are always 1, solution of KPBS. However, as the number of edges in J' depends on the weights of the edges of J the algorithm is only pseudo-polynomial and therefore not useful in practice.

Theorem 2 proves that the multigraph algorithm is a $\frac{8}{3}$-approximation algorithm. Before that we need the following lemma. It proves that the lower bound of the useful transmission time of graph I is equal to $\max\left(\left\lceil \frac{P(H)}{k} \right\rceil, W(H)\right)$.

Lemma 2: $\eta_d(I) = \max\left(\left\lceil \frac{P(H)}{k} \right\rceil, W(H)\right)$

Input: A bipartite graph $G = (V_1, V_2, E, w_G)$, an integer k.
Output: A set of matchings S.

1. Build the graph J as described in steps 1, 2 and 3 of GGP
2. Build $J' = trans(J)$
3. $S' = \emptyset$
4. While $E_{J'} \neq \emptyset$ do:
   5.1. Choose a perfect matching M' in J' with $\forall e \in M w_M(e) = 1$
   5.2. Add M' to S', the set of solution matchings
   5.3. Remove from $E_{J'}$ all edges of M'
6. Remove all edges e in S' such that $e \notin E$

Fig. 8. **Multigraph algorithm**

Proof of Lemma 2:: Consider the 3 possible cases when building I:

1) $\frac{P(H)}{k} \geq W(H)$ and $\frac{P(H)}{k} \in \mathbb{N}$: $I = H$ and therefore $\eta_d(I) = \max\left(\frac{P(I)}{k}, W(I)\right) = \max\left(\frac{P(H)}{k}, W(H)\right) = \max\left(\left\lceil \frac{P(H)}{k} \right\rceil, W(H)\right)$.
2) $\frac{P(H)}{k} \geq W(H)$ and $\frac{P(H)}{k} \notin \mathbb{N}$: we add no edge of weight greater than $W(H)$ and $\frac{P(I)}{k} = \left\lceil \frac{P(H)}{k} \right\rceil$ hence $\eta_d(I) = \max\left(\left\lceil \frac{P(H)}{k} \right\rceil, W(H)\right)$.
3) $\frac{P(H)}{k} < W(H)$: as we add no edge of weight greater than $W(H)$, $W(I) = W(H)$. Moreover we only add edges until $\frac{P(I)}{k} = W(H)$ d'o $\eta_d(I) = \eta_d(H) = W(H)$.

∎

Theorem 2: The multigraph algorithm is a $\frac{8}{3}$-approximation algorithm.

Proof of Theorem 2:: With as input parameters $G = (V_1, V_2, E, w_G)$ a bipartite graph and k an integer, we apply the multigraph algorithm on G to obtain S' the set of wanted matchings. J is, by construction, a $\frac{P(I)}{k}$-weight-regular graph and all of its weights are integers, therefore $J' = trans(J)$ is a $\frac{P(I)}{k}$-regular multigraph. Since $P(I) = m(I')$, J' is a $\frac{m(I')}{k}$-regular graph, where $I' = trans(I)$. As at each step of the main iteration we remove a perfect matching from J', $|S'| = \frac{m(I')}{k} = \eta_s(I')$. The cost of the solution S' is therefore $c(S') = \eta_s(I') + \eta_s(I')$ because each step has a duration of 1 and the startup delay β is considered equals to 1.Therefore:

$$c(S') = 2\eta_s(I') \tag{1}$$

If all edges of G have a weight less than 1 then all edges of H have a weight of 1 and $W(H) = 1$. Hence, all edges added to H to build I have a weight of 1. This means that $\eta_s(I') = \eta_s(I)$ because I and I' are then identical graphs. As the weight of any edge in I is 1, we have $P(I) = m(I)$ and $W(I) = \Delta(I)$ and therefore: $\eta_s(I) = \max\left(\left\lceil \frac{P(I)}{k} \right\rceil, W(I)\right) = \max\left(\left\lceil \frac{P(H)}{k} \right\rceil, W(H)\right)$ by construction of I. As the weight of each edge in H is

also 1, we can conclude that $\eta_s(I) = \eta_s(H)$. Finaly as $m(H) = m(G)$ and $\Delta(H) = \Delta(G)$ we have: $\eta_s(I') = \eta_s(I) = \eta_s(H) = \eta_s(G)$ and therefore equation (1) becomes $c(S') = 2\eta_s(G) \leq 2\eta(G)$. Therefore the algorithm is a 2-approximation algorithm when all edges of G have a weight less than 1.

As $\frac{P(I)}{k} = \frac{m(I')}{k}$ and $W(I) = \Delta(I')$ we know that $\eta_s(I') = \max\left(\Delta(I'), \frac{m(I')}{k}\right) = \max\left(W(I), \frac{P(I)}{k}\right) = \eta_d(I)$. Consequently $c(S') = 2\eta_d(I)$. We now use Lemma 2 to deduce that

$$c(S') = 2\max\left(\left\lceil\frac{P(H)}{k}\right\rceil, W(H)\right) \quad (2)$$

Let us first suppose that $\left\lceil\frac{P(H)}{k}\right\rceil > W(H)$.

The equation (2) becomes: $c(S') = 2\left(\left\lceil\frac{P(H)}{k}\right\rceil\right)$.

In the algorithm building H from G, no edge sees its weight increasing by more than one unit. Therefore, we have: $P(H) \leq P(G) + m(G)$. This leads to:

$$\begin{aligned} c(S') &\leq 2\left(\left\lceil\frac{P(G)+m(G)}{k}\right\rceil\right) && (3)\\ &\leq 2\left(\left\lceil\frac{P(G)}{k}\right\rceil + \left\lceil\frac{m(G)}{k}\right\rceil\right)\\ &\leq 2\left(\left\lceil\frac{P(G)}{k}\right\rceil + \max\left(\left\lceil\frac{m(G)}{k}\right\rceil, \Delta(G)\right)\right) && (4)\\ &\leq 2\left(\frac{P(G)}{k} + 1 + \eta_s(G)\right)\\ &\leq 2\left(\eta_d(G) + 1 + \eta_s(G)\right) = 2\eta(G) + 2 && (5) \end{aligned}$$

Since $W(H)$ is an integer and we supposed that $\left\lceil\frac{P(H)}{k}\right\rceil > W(H)$ we can deduce that $\frac{P(H)}{k} > W(H)$. This means that $m(H) > k$ (otherwise, the number of edges of H would be greater that k then $P(H)$ would be less than $k \times W(H)$ – by definition of $W(H)$ – and consequently $\frac{P(H)}{k}$ would be less than $W(H)$).

By construction $m(H) = m(G)$ therefore $\left\lceil\frac{m(G)}{k}\right\rceil \geq 2$. We deduce that $\eta_s(G) \geq 2$.

We can assume that the weight of at least one edge of G is greater than 1 or equal to 1 (the case when all the edges have a weight smaller than 1 has been treated above and leads to an approximation ratio of 2). We have $W(G) \geq 1$ and therefore $\eta_d(G) = \max\left(\frac{P(G}{k}, W(G)\right) \geq 1$. Consequently $\eta(G) \geq 2 + 1 = 3$.

Inequation (3) becomes

$$\frac{c(S')}{\eta(G)} \leq 2 + \frac{2}{\eta(G)} \leq 2 + \frac{2}{3} = \frac{8}{3}$$

This means that in this case the algorithm is a $\frac{8}{3}$-approximation algorithm.

Now we still have to study the case where $\left\lceil\frac{P(H)}{k}\right\rceil \leq W(H)$. In this case equation (2) becomes: $c(S') = 2W(H)$.

However, $W(H) \leq W(G) + \Delta(G)$ because we have added at most one to each edge weight of G to build H. Hence,

$$\begin{aligned} c(S') &\leq 2\left(W(G) + \Delta(G)\right)\\ &\leq 2\left(\max\left(\frac{P(G)}{k}, W(G)\right)\right.\\ &\quad \left.+\max\left(\left\lceil\frac{m(G)}{k}\right\rceil, \Delta(G)\right)\right)\\ &\leq 2\left(\eta_d(G) + \eta_s(G)\right) = 2\eta(G) \end{aligned}$$

Therefore for this case the approximation ratio is 2.

Consequently the approximation ratio for the multigraph algorithm is $\frac{8}{3}$ (in the worst case). ∎

From the above theorem it follows that GGP is a $\frac{8}{3}$-approximation algorithm. Indeed, for any schedule S obtained by GGP of cost $c(S)$ it exists a schedule S' obtained by the multigraph algorithm of cost $c(S')$ such that $c(S) \leq c(S')$. By construction, a solution S obtained by GGP can be decomposed into a solution S' where $\forall M_i \in S$ of cost $c(M_i)$ there exists $c(M_i)$ identical matchings M'_j of cost 1 in S'. We therefore have $\sum_{i=1}^{|S|} c(M_i) = \sum_{j=1}^{|S'|} c(M'_j)$ and $|S| \leq |S'|$.

The cost of S is: $|S| + \sum_{i=1}^{|S|} c(M_i)$. Similarly the cost of S' is: $|S'| + \sum_{j=1}^{|S'|} c(M'_j)$. Therefore $c(S) \leq c(S')$.

3) Complexity: We have shown that the Multigraph algorithm is pseudo-polynomial. Here we show that GGP is polynomial and compute its worst-case complexity.

Proposition 3: GGP has a worst case complexity in $O(\sqrt{n(G)}(m(G) + n(G))^2)$.

Proof of Proposition 3: Steps 1,2,3,4 and 6 of GGP are computed in linear time. However steps 5 requires finding a perfect matching which is done in $O(\sqrt{n(J)}m(J))$ using the hungarian method [22]. At each step we remove at least one edge (the edge of the matching with the lowest weight) hence we iterate at most $m(J)$ times. Therefore the worst case complexity of step 5 is in $O(\sqrt{n(J)}m(J)^2)$.

Now to build H no edge or node are added, hence $n(H) = n(G)$ and $m(H) = m(G)$. To build I we add at most k edges and therefore $2k$ nodes hence $n(I) \leq n(G) + 2k$ and $m(I) \leq m(G) + k$. Since it has no sense allowing to select in a matching more edges than nodes, we limit here k to $n(G)$. We can then deduce that $n(I) \leq 3n(G)$ and $m(I) \leq m(G) + n(G)$.

In the weight-regular extension algorithm, for each node in I (each iteration) we add at most one new node in J. Therefore $n(J) \leq 2n(I)$. Similarly for each node in I we add at most 2 new edges in J and therefore $m(J) \leq m(I) + 2n(I)$. Hence, $n(J) \leq 6n(G)$ and $m(J) \leq m(G) + 7n(G)$.

This leads to a worst case complexity of step 5 of $O(\sqrt{n(G)}(m(G) + n(G))^2)$. ∎

C. OGGP

1) Algorithm: It is possible to find a family of graphs on which GGP reaches a 2-approximation ratio. We developed a modified version of GGP called OGGP which gives good results on this family of graphs, and better results than GGP in the general case. Being a direct extension of GGP, OGGP inherits the $\frac{8}{3}$-approximation ratio. It should be noted however, that we have no proof that OGGP could have a lower approximation ratio than $\frac{8}{3}$.

The principle is the following. In GGP, when choosing a perfect matching, the weight-regular graph guarantees that there exists a perfect matching. However, there is often more than one perfect matching and GGP doesn't specify which one to choose, but uses a random one. In OGGP we simply try to choose the matching that might give the best results among all matchings. Intuitively, we would like to issue as much communications as possible. By taking the longest possible communication steps, we might reduce the total number of steps, and therefore the communication time. A communication step time is given by the smallest weight of all edges in the matching. To have the largest communication step, we need to find the perfect matching whose smallest weight is maximal.

The greedy algorithm depicted in Fig 9 finds a perfect matching which smallest edge's weight is maximal. It is based on the algorithm described in [4] that maximizes the minimum weight of a matching.

Input: A bipartite graph G.
Output: M: the perfect weighted matching with maximal minimum weight.

1. $G' = \emptyset, M = \emptyset, G'' = G$
2. while M is not perfect in G do:
   2.1. choose $e \in E(G'') | \forall e' \in E(G''), w(e) \geq w(e')$
   2.2. $E(G') = E(G') \cup e$
   2.3. $E(G'') = E(G'') \backslash e$
   2.4. M = a maximal matching in G'

Fig. 9. **Algorithm for extracting a matching with maximal minimum weight**

Proposition 4: Algorithm of Fig 9 returns a matching of maximal minimum weight.

a) Proof of Proposition 4: Let l be the last edge added at the previous step in G', we have $l \in M$ because without l it was not possible to find a perfect matching in G. We also have $\forall e \in G, w(e) > w(l) \Rightarrow e \in G'$. Suppose that M' is a maximal matching better than M (it's minimum weight is larger than the one of M). M' is such that: $\forall e \in M', f(e) > f(l)$. Therefore, we have: $M' \subset G'$. This is a contradiction, hence M is a perfect matching maximizing the minimum weight. ∎

2) Complexity: The new matching algorithm complexity is $O(m(J)\sqrt{n(J)}m(J)) = O(m(J)^2\sqrt{n(J)})$. Therefore, the complexity of OGGP is $O(\sqrt{n(G)}(m(G) + n(G))^3)$.

VII. Heuristics

Here are two heuristics that appear to work well in practice (a heuristic on weights and a heuristic on degrees). They are faster than GGP and OGGP but are not approximation algorithms as will show the simulation results. The heuristic on degrees is the same as the heuristic on weights except that line 2. is changed into "2. *Keep only the k (or less if there are less than k edges) edges with highest degrees.*".

Input: A weighted bipartite graph G, an integer k.
Output: A set of valid weighted matchings.

1. Find a maximal matching.
2. Keep only the k (or less if there are less than k edges) edges whose weights are the largest.
3. Set all the weights of the matching equal to the lowest one.
4. Subtract the matching from G.
5. Loop until there is no more edge left in G.

Fig. 10. **Heuristic on weights**

Complexity:: We use the Hungarian method of complexity $\mathcal{O}(m(G)n(G)^{1/2})$ for finding a maximum cardinality matching in a bipartite graph. For both heuristics, at each step, at least one edge is removed from G. Therefore, the complexity of both heuristics is $\mathcal{O}(m(G)^2\sqrt{n(G)})$ which is better than the complexity of GGP.

VIII. Experiments

A. Simulation of the Heuristics

We have tested each heuristic (with k fixed) on a sample of 100 000 random graphs (the number of edges, the edges, and finally the weights were chosen randomly with a uniform distribution) with 20 nodes on each side. We made a difference between lightly and heavily weighted graphs. Small weights were taken between 1 and 20, whereas large weights were taken between 1 and 100 000. The result of a heuristic is calculated as the solution cost divided by the lower bound η. We call this ratio the *evaluation ratio*.

In Fig 11, 12, 13 and 14 the plots show the average and the maximum calculated over the samples.

For these tests, the maximum is always below 2.4, with an average under 1.8 for small weights, and below 2, with an average under 1.3 in case of large weights.

We explain the convex shape of the plots as follows:

- when $k = 1$ the two heuristics obtain the optimal solution which consists in one communication per step;
- when k is greater than 2 and lower than a certain value (close to $n/2$), the quality of the solution degrades (compared to the lower bound); We believe that this is due to the fact that, at each step, the number of valid matchings increases;
- When k is greater than $n/2$ the quality of the solution tends to improve. At each stage of the two heuristics the choice of valid matchings decreases, therefore the heuristics are less likely to select bad valid matchings.

B. Simulation of GGP and OGGP

The simulation of GGP and OGGP has been conducted under the same conditions as the simulation of the heuristics. OGGP and GGP have been implemented into a C++ library, and executed on random graphs as described in the previous section.

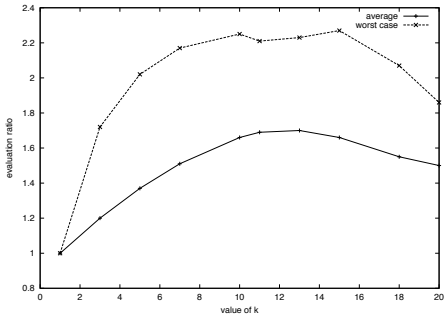


Fig. 11. Heuristic on weights. Simulation with small weights.

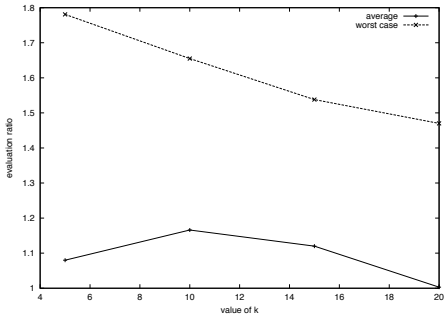


Fig. 12. Heuristic on weights. Simulation with large weights.

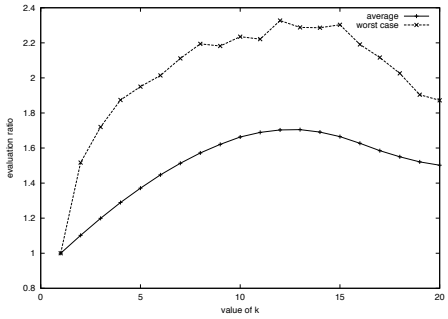


Fig. 13. Heuristic on edges. Simulation with small weights.

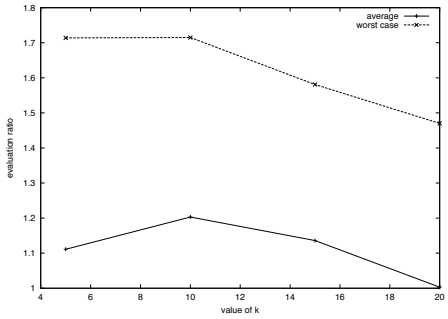


Fig. 14. Heuristic on edges. Simulation with large weights.

1) Comparing GGP and OGGP:

a) Tests on Small Weights.: Fig 15 displays how the evaluation ratio varies when k grows. The weights are generated randomly between 1 and 20.

We can see that as k grows the evaluation ratio grows and stabilizes. OGGP gives better results than GGP even when comparing the worst case obtained with OGGP and the average obtained with GGP.

b) Tests on Large Weights.: Fig 15 displays how the evaluation ratio varies when k grows. The weights are generated randomly between 1 and 100000.

Results are similar but with an evaluation ratio far closer to 1 on large weights. On these cases, the difference between GGP and OGGP is smaller.

We can see that GGP is giving better results than the heuristics. Although the difference is not extremely high on average, when comparing worst cases, heuristics are taking 1.5 more time than GGP.

C. Real-World Experiments with MPI

In order to validate theoretical results and simulations, we have conducted several real-world experiments. We used two clusters of 10 1.5 GHz Pentium computers running Linux. Network cards were 100Mbits Ethernet adapters and the two clusters were interconnected within a local network by two 100Mbits switches. In order to test interesting cases, that is where $k \neq 1$, we limited the available incoming and outgoing bandwidth of each network card to $\frac{100}{k}$ Mbits per second. This was done using the *rshaper* [29] Linux kernel module. This module implements a software token bucket filter thus enabling a control of the available bandwidth. We conduct experiments for $k = 3, k = 5, k = 7$.

Two different types of redistribution have been implemented. First, a brute force TCP-only approach: we start all communications simultaneously and wait until all transfers are finished. In this case the network transport layer (TCP) is responsible for the congestion control. The second approach allows us to test our algorithms: we divide all communications into different steps, synchronized by a barrier, and only one synchronous communication can take place in each step for each sender. Both algorithms have been implemented using MPICH version 1.2.4. We have not implemented an exponential algorithm finding the optimal solution (which could seem possible as the number of nodes and edges is not very high) because designing such an algorithm is difficult, and anyway our algorithms are already close enough to the optimum. All communication times have been measured using the *ntp_gettime* function call from the GNU libc.

In our tests, the 10 nodes of the first cluster have to communicate to each 10 nodes of the second cluster. The size of the data to transfer between two given cluster nodes is uniformly generated between 10 and n MB. We plot the total communication time obtained when n increases as shown in Fig. 16.

Several observations can be made:

- We achieve a 5% to 20% reduction of communications costs. Although we are alone on a local network, where TCP is efficient, we are able to achieve better results.
- The barriers cost extremely little time. Although OGGP algorithm has 50% less steps of communication, it gives the same result as GGP. However we believe the cost of synchronizations may increase if we introduce some random perturbations on the network.
- The brute-force approach does not behave deterministi-

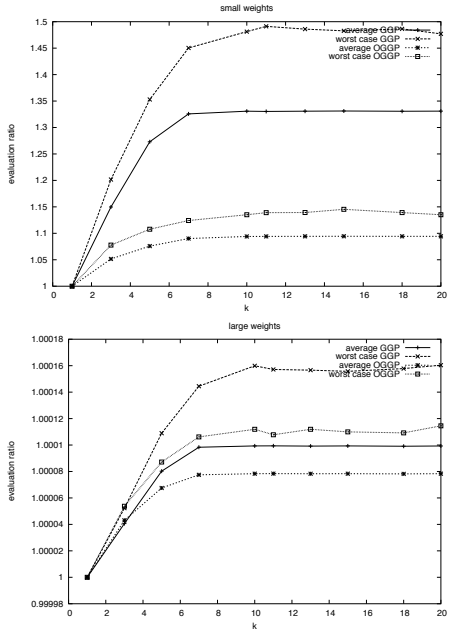


Fig. 15. GGP and OGGP for Small Weights / Large Weights graphs

cally. When conducting several time the same experiments we see a time variation of up to 10 percents. It is interesting to see that our approach on the opposite behaves deterministically.

- As the available bandwidth decreases (i.e. k increases) we increase the benefits of using GGP or $OGGP$ over the brute-force approach.

IX. Conclusions

In this paper we have formalized and studied the problem (called KPBS) of redistributing parallel data over a backbone. Our contribution is the following: We have shown that KPBS remains NP-hard when k is constant. We have studied lower bounds related to KPBS. We have proposed a polynomial time approximation algorithm with ratio 2. We have then proposed an improvement of this algorithm. Two messages scheduling algorithms called GGP and OGGP for the redistribution problem with a lower complexity have been proposed. GGP and OGGP provide a solution at most twice longer than the optimal. We then studied two fast heuristics. Simulations show that OGGP outperforms GGP that outperforms the heuristics. We have performed real experiments on two clusters. Results show that our scheduling algorithms outperform the brute-force approach that consists in letting the network manage the congestion alone (redistribution time can be reduced to up to 20%).

In our approach we limit the maximum number of messages during one step. This is especially useful when the redistribution is performed between two clusters interconnected by a backbone and when this backbone is a bottleneck. However the algorithms we have proposed can also be used when a redistribution occurs on the same parallel machines or in the context of SS/TDMA systems or WDM network.

In our future work, we want to extend the model to handle more complex redistributions. First we would like to consider

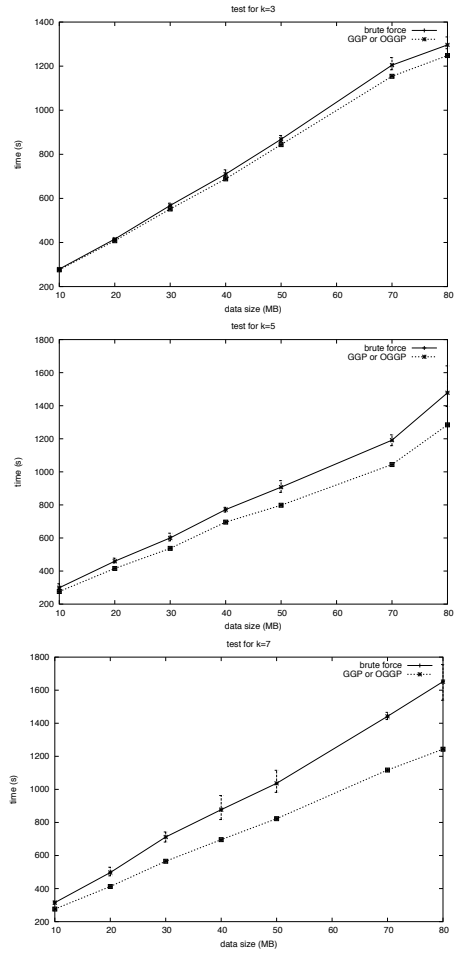


Fig. 16. *Brute-Force vs. GGP or OGGP ($k = 3, 5, 7$)*

achieving a local pre-redistribution in case a high-speed local network is available. This would enable to aggregate small communications together, or on the opposite to dispatch communications to all nodes in the cluster. Second, we would like to study the problem when the throughput of the backbone varies dynamically or when the redistribution pattern is not fully known in advance. We think that our multi-step approach could be useful for these dynamic cases. The final goal of this work is to produce (together with the people involved in the INRIA ARC redGRID [4]) a fully working redistribution library.

References

[1] F. N. Afrati, T. Aslanidis, E. Bampis, and I. Milis. Scheduling in switching networks with set-up delays. *J. Comb. Optim.*, 9(1):49–57, 2005.

[2] F. Bertrand, R. Bramley, D. Bernholdt, J. A. Kohl, A. Sussman, J. W. Larson, and K. Damevski. Data redistribution and remote method invocation in parallel component architectures. In *IPDPS*, 2005.

[3] P. B. Bhat, V. .K. Prasanna, and C. .S. Raghavendra. Block Cyclic Redistribution over Heterogeneous Networks. In *11th International Conference on Parallel and Distrinuted Computing Systems (PDCS 1998)*, 1998.

[4] G. Bongiovanni, D. Coppersmith, and C. K. Wong. An Optimum Time Slot Assignment Algorithm for an SS/TDMA System with Variable Number of Transponders. *IEEE Trans. Comm.*, 29(5):721–726, 1981.

[4] `http://graal.ens-lyon.fr/~desprez/REDGRID`

[5] V. Boudet, F. Desprez, and F. Suter. One-step algorithm for mixed data and task parallel scheduling without data replication. In *IPDPS*, page 41, 2003.

[6] H. Casanova and J. Dongarra. NetSolve: A Network-Enabled Server for Solving Computational Science Problems. *International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212 – 213, Fall 1997.

[7] H. Choi, H.-A. Choi, and M. Azizoglu. Efficient Scheduling of Transmissions in Optical Broadcast Networks. *IEEE/ACM Trans. Net.*, 4(6):913–920, 1996.

[8] P. Crescenzi, D. Xiaotie, and C. H. Papadimitriou. On Approximating a Scheduling Problem. *Journal of Combinatorial Optimization*, 5:287–297, 2001.

[9] B. Del-Fabbro, D. Laiymani, J.-M. Nicod, and L. Philippe. Data management in grid applications providers. In *DFMA*, pages 315–322, 2005.

[10] F. Desprez, J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert. Scheduling Block-Cyclic Array Redistribution. *IEEE TPDS*, 9(2):192–205, 1998.

[11] F. Desprez and E. Jeannot. Improving the GridRPC Model with Data Persistence and Redistribution. In *3rd International Symposium on Parallel and Distributed Computing (ISPDC)*, Cork, Ireland, July 2004.

[12] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problem. *SIAM J. Comput.*, 5:691–703, 1976.

[13] A. Ganz and Y. Gao. A Time-Wavelength Assignment Algorithm for WDM Star Network. In *IEEE INFOCOM'92*, pages 2144–2150, 1992.

[14] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the theory of NP-Completeness*. W.H Freeman and co., 1979.

[15] I. S. Gopal, G. Bongiovanni, M. A. Bonuccelli, D. T. Tang, and C. K. Wong. An Optimal Switching Algorithm for Multibean Satellite Systems with Variable Bandwidth Beams. *IEEE Trans. Comm.*, COM-30(11):2475–2481, November 1982.

[16] I.S. Gopal and C.K. Wong. Minimizing the Number of Switching in an SS/TDMA System. *IEEE Trans. Comm.*, 1985.

[17] M. Guo and I. Nakata. A framework for efficient data redistribution on distributed memory multicomputers. *The Journal of Supercomputing*, 20(3):243–265, 2001.

[18] S. Sekiguchi H. Nakada, M. Sato. Design and Implementations of Ninf: Towards a Global Computing Infrastructure. *Future Generation Computing Systems, Metacomputing Issue*, 15:649–658, 1999.

[19] C.-H. Hsu, Y.-C. Chung, D.-L. Yang, and C.-R. Dow. A generalized processor mapping technique for array redistribution. *IEEE TPDS*, 12(7):743–757, 2001.

[20] The hydrogrid project. `http://www-rocq.inria.fr/~kern/HydroGrid/HydroGrid-en.html`.

[21] Oak Ridge National Labs. Mxn. `http://www.csm.ornl.gov/cca/mxn`.

[22] S. Micali and V. V. Vazirani. An $o(\sqrt{(}v)e)$ algorithm for finding a maximum matching in general graphs. In *Proc. 21st Ann IEEE Symp. Foundations of Computer Science*, pages 17–27, 1980.

[23] M. Mishra and K. Sivalingam. Scheduling in WDM Networks with Tunable Transmitter and Tunable Receiver Architecture. In *NetWorld+Interop Engineers Conference*, Las Vegas, NV, May 1999.

[24] N. Park, V. K. Prasanna, and C. S. Raghavendra. Efficient algorithms for block-cyclic array redistribution between processor sets. *IEEE TPDS*, 10(12):1217–1239, 1999.

[25] C. Pérez, T. Priol, and A. Ribes. A parallel corba component model for numerical code coupling. In *Proc. of the 3rd International Workshop on Grid Computing*, number 2536 in LNCS, pages 88–99, Baltimore, Maryland, USA, November 2002.

[26] G. R. Pieris and Sasaki G.H. Scheduling Transmission in WDM Broadcast-and-Select Networks. *IEEE/ACM Transaction on Networking*, 2(2), April 1994.

[27] A. Radulescu, C. Nicolescu, A. J. C. van Gemund, and P. Jonker. CPR: Mixed Task and Data Parallel Scheduling for Distributed Systems. In *IPDPS*, page 39, 2001.

[28] N. Rouskas and V. Sivaraman. On the Design of Optimal TDM Schedules for Broadcast WDM Networks with Arbitrary Transceiver Tuning Latencies. In *IEEE INFOCOM'96*, pages 1217–1224, 1996.

[29] A. Rubini. Linux module for network shaping `http://ar.linux.it/software/\#rshaper`.

[30] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. ACM Press, New-York, 1999.

[31] J. Turek, J. Wolf, and P. Yu. Approximate Algorithms Scheduling Parallelizable Tasks. In *Proceedings of the fourth annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'92)*, pages 323–332. ACM Press, 1992.

Johanne Cohen Johanne Cohen received her PhD in computer science from the University of Paris XI in 1999. She is currently at the French Centre National de la Recherche Scientifique (CRNS) and located at LORIA Laboratory (Nancy, France). Her main research interest include the study of telecommunication networks and approximation methods for hard combinatorial problems.

Emmanuel Jeannot Emmanuel Jeannot is currently full-time researcher at INRIA (Institut National de Recherche en Informatique et en Automatique) and is doing its research at theLORIA laboratory. From sept. 1999 to spet. 2005 he was associate professor at the Université Henry Poincaré, Nancy 1. He got his PhD and Master degree of computer science (resp. in 1996 and 1999) both from Ecole Normale supérieure de Lyon. His main research interests are scheduling for heterogeneous environments and grids, data redistribution, grid computing software, adaptive online compression and programming models.

Nicolas Padoy Nicolas Padoy studied computer science at the Ecole Normale Suprieure de Lyon, France, and at the Technische Universitt Mnchen, Germany. After some research in distributed computing he is now starting a joint PhD in medical computer vision at the Universit Henry-Poincar, France, and at the Technische Universitt Mnchen, Germany.

Frédéric Wagner Frederic Wagner has been studying computer science at the Université Louis Pasteur in Strasbourg. After a master degree on automatic compiler optimizations he is now finishing his PhD in Nancy at Université Henri Poincaré on communications scheduling for parallel computing.

Improving Middleware Performance with AdOC: an Adaptive Online Compression Library for Data Transfer

Emmanuel Jeannot
LORIA, Université H. Poincaré Nancy - 1
Vandœuvre les Nancy, France
Emmanuel.Jeannot@loria.fr

Abstract

In this article, we present the AdOC (Adaptive Online Compression) library. It is a user-level set of functions that enables data transmission with compression. The compression is performed dynamically during the transmission and the compression level is constantly adapted according to the environment. In order to ease the integration of AdOC into existing software the API is very close to the read and write UNIX system calls and respects their semantic. Moreover this library is thread-safe and is ported to many UNIX-like systems. We have tested AdOC under various conditions and with various data types. Results show that the library outperforms the POSIX read/write system calls on a broad range of networks (up to 100 Mbit LAN), whereas on Gbit Ethernet, it provides similar performance.

1. Introduction

Computational and data grids are distributed architectures that interconnect a set of heterogeneous computers (from a parallel machine to a desktop PC) with various types of networks (Internet, WAN, LAN, etc.). The objective of such grids is to gather distributed resources (CPU, disk, memory, etc.), to solve problems that require huge amount of computation or storage. Nowadays many middlewares [5, 6, 9, 17] are under development to allow applications to use grids in a transparent way. These middlewares manage the infrastructure, schedule the jobs, handle communications and data. In order to do this each middleware has to rely on a set of services (scheduling, accounting, resource discovery, etc.). In this paper we propose and describe a new service for grid middlewares and data transfer tools that enables compression on the fly for efficient transmission. The motivation for this work is that many (grid) applications require a large amount of data to be transmitted. In some cases, data transmission is the most time consuming part and therefore needs to be optimized.

The service we propose here is a library called AdOC (Adaptive Online Compression), which offers the possibility to transfer data while compressing it. It is an *adaptive* service as the compression level is dynamically changed according to the environment and the data. The adaptation process is required by the heterogeneous and dynamic nature of grids. For instance if the network is very fast, time to compress the data may not be available. But, if the visible bandwidth decreases (due to some congestion on the network), some time to compress the data may become available.

In this paper, AdOC is tested on several kind of networks and with different types of data. We compare AdOC read and write functions to the standard POSIX read and write system calls. We show that the latency of AdOC is similar to that of the POSIX read/write. We show that AdOC enables an increase of bandwidth depending on the data sent and the network (up to 6 times faster). We provide a conservative approach of compression that leads to no performance degradation on most kinds of networks (on Gbit Ethernet some microseconds are lost with AdOC) and any kind of data (even an incompressible one).

The API of the AdOC library is very close to the POSIX read/write system calls and respects their semantic. Therefore, it has been easily integrated into the NetSolve middleware [6]. The evaluation of the enhanced version of NetSolve shows a significant increase of performance on various scenarii whereas, on worst-case scenarii, no performance degradation is seen.

2. Adaptivity issues

Compression is often proposed in various transmission protocols such as FTP [15], PPP [16] or in the secure copy tool (`scp`). However, compressing is never the default behavior because no adaptation is provided. In some cases, it is worth to compress but not always.

In this paper, adapting means changing the compression level during the transmission. The *compression level* refers

on how efficiently data are compressed. Adapting the compression level (and in some cases disabling the compression) must be performed according to the following parameters:

- *Current speed of the network*. If the network is very fast, there is no time to compress the data. If the network is slow enough, some time may be available to compress the data. Moreover, the network is often shared by other users. Thus, its speed can change with the time: it is then required to change the compression level.
- *Current speed of the machine* on each side of the transmission. Compressing and uncompressing data requires some computational power. Before enabling compression, one must be sure that machines in both ends have enough computational power to perform the compression/decompression, without slowing down the transmission. Indeed, if it requires more time to compress, send and uncompress the data than just send the data uncompressed no gain can be expected. Moreover, many machines run multi-task operating systems (for instance UNIX). Therefore, the available CPU power may change with the time. In this case, it is required to adapt the compression level to the new conditions.
- *Size of the data to be transmitted*. Enabling compression adds a startup time (latency) to the transmission. Therefore, if small messages are to be sent, the startup time can be greater than the gain obtained with compression. Hence, for small data, the compression must be disabled.
- *Type of the data to be transmitted*. Some data are easier to compress than other. ASCII data compresses better and requires less time to compress than binary data. Moreover, for some files (such as directories archive), the nature of the data changes along the file. Hence, the compression level must be constantly adapted to the type of the data.

The compression level adaptation must be performed according to all these parameters at the same time. For instance, we have to take into consideration the ratio between the available bandwidth and the CPU power more than each criteria separately.

Table 1 shows compression timings. Two same size files have been compressed using either gzip [8] or lzf [13] tools on a 1 GHz PowerPC G4 under MacOS X 10.2.8. `oilpann.hb` is a sparse matrix file in the Harwell-Boeing format (ASCII). `bin.tar` is a tarball of executables. Lzf and gzip as well as their related libraries (liblzf and zlib) provide lossless compression based on the Ziv-Lempel algorithm [20, 21]. We see that lzf is a fast compression algorithm with low compression ratio. Concerning gzip, we see that the compression time (columns *c. time*) increases with the compression level as the decompression time (columns *d. time*) is roughly constant. After level 6 the compression ratio (columns *ratio*) does not increase significantly.

	`oilpann.hb`			`bin.tar`		
algo	c. time	ratio	d. time	c. time	ratio	d. time
lzf	1.5	3.26	2.7	2.3	1.68	3.2
gzip 1	4.4	4.88	2.7	8	2.23	3.1
gzip 2	4.4	5.13	3	8.6	2.27	3.3
gzip 3	4.6	5.52	3	10	2.31	3.1
gzip 4	6	5.83	2.5	11.5	2.38	2.9
gzip 5	6.6	6.32	2.9	12.3	2.43	3
gzip 6	8.1	6.64	2.5	16.3	2.44	3
gzip 7	10.1	6.75	2.8	18.4	2.45	3.5
gzip 8	26.7	6.99	3.8	24.1	2.45	3
gzip 9	46	7.02	2.6	34.3	2.46	3.2

Table 1. Compression Timings on Bench Files Using lzf and Different Levels of gzip

For some specific data it may happen that the size of the compressed data is larger than the size of the uncompressed data. This is the case for already compressed data. In this case, tools like gzip [8] guarantee that the size does not increase more than 0.0015% for such files.

In this paper, compression level 0 will mean no compression (no time is used to compress the data). For compression level 1 we will use lzf, for compression level 2 we will use gzip at level 1, ...

3. The AdOC Algorithm

3.1. Principle

The AdOC algorithm has been proposed by Jeannot, Knutsson and Bjorkman in [11]. It is a general-purpose user-level and portable algorithm suited not only for grid computing but also for any data transfer application. It is mainly based on two ideas:

- Compression and communication overlap. When a process performs some I/O (such as accessing a disk or a network socket) it is blocked until the device becomes ready. During that time, the processor is available to perform some computation. Overlapping compression with communication allows the compression time to become mostly invisible to the user. We also perform decompression and communication overlap on the receiver side for the same reason.

- Dynamic adaptation of the compression level. We saw in the previous section that the compression time depends on the compression level. Moreover, the environment (CPU/network speed, data, etc.) is subject to change with the time. Therefore, the available time to compress/decompress data changes during the data transfer. We adapt to the change of the environment by changing the compression level.

The AdOC algorithm is presented Figure 1, and works as follows. It uses:

- Multithreading. The sending process is made of two threads. One thread compresses the data. The other one sends the data on the network. On the receiving side the process is also made of two threads. One reads the network the other one decompresses the data. Multithreading allows to overlap the compression/decompression and the communication.

- FIFO queues. A queue is used to store data shared by the threads. On the sending side, the compression thread stores data in the queue, the emission thread reads this data and sends it to the network. On the receiving side, the reception thread reads the network and stores the data into the queue, the decompression threads reads the data from the queue to decompress it.

3.2. The compression thread

The compression thread has in charge to compress either a file or an array of bytes. In order to do that, it splits the file or the array into chucks of fixed size called *buffers*. The compression level is updated before reading a new buffer. Therefore, a tradeoff has to be found for the buffer size. If the size is too large, the reactivity needed to adapt the compression level may not be good enough. If the size is too small, the total amount of data sent will increase. Indeed, due to internal data structures of compression algorithms, compressing a file at a given level leads to a smaller compressed file than splitting the file, compressing each part and merging the compressed parts. In our implementation, the size of each buffer is chosen to be 200 KB. For this size, less than 6% of compression degradation is seen and the reactivity appears to be good enough [11].

Once the compression level is updated, a buffer is compressed at this level. Each time a *packet* of compressed data is generated, this packet is read and stored in the FIFO queue and the compression is resumed. In our implementation, the size of a packet is 8KB. If the compression is disabled, (compression level = 0) only an uncompressed packet is stored in the queue and the compression level is updated.

```
Input  n: number of packets in the queue
       δ: variation of the size of the queue
       l: old compression level
Output l: new compression level

1. if n=0
2.   return minLevel
3. if n< 10
4.   if δ ≤ 0
5.       l=l/2
6. else if n< 20
7.   if δ > 0
8.       l++;
9.   else if (δ < 0)
10.      l−−;
11. else if n< 30
12. if δ > 0
13.      l+=2;
14. else if δ < 0
15.      l−−
16. else if δ > 0
17. l+=2
18. l=max(l,minLevel)
19. l=min(l,maxLevel)
20. return l;
```

Figure 2. *Compression Level Update Algorithm*

3.3. Adapting the compression level with the FIFO queue

The AdOC algorithm monitors the size of the FIFO queue on the emission side as well as the variation of its size. The size of the queue is the number of stored packets. This information is used to update the compression level as shown in Fig. 2. The idea is the following:

- If the size of the queue increases, this means that the network and the receiver consume data slower than it is produced by the compression thread. Some extra time is therefore available for compression: the compression level is then increased.
- If the size of the queue decreases, this means that the network and the receiver consume data faster than it is produced by the compression thread. It is required to decrease the compression level in order to generate packets at a greater rate.

The goal of changing the compression level is to avoid the queue to become either empty or too large. If the queue

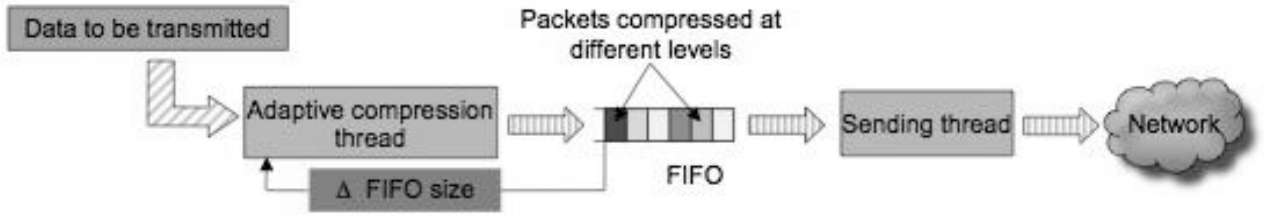


Figure 1. AdOC Algorithm: Emission Process (Reception Process is Symmetric but does not Monitor the Queue Size)

becomes empty, this means that the sending thread is waiting for data to be sent and therefore the transmission is slowed down. In order to avoid this to happen, some thresholds are added as describe in Figure 2. The compression level cannot increase if the queue size is too small (less than 10 packets). The level is increased by 2 (resp. divided by 2) if the queue is very large (resp. very small).

We see that the AdOC algorithm has a conservative strategy. As each packet has a size of 8 KB, and no compression is performed before the size of the FIFO becomes larger than 10 packets, no compression is done for data smaller than 80 KB.

4. AdOC Library

The AdOC library is an implementation of the AdOC algorithm. This library provides a set of user-level functions to send and receive data through sockets. The main features of this library are: synthetic API, full respect of the read/write UNIX system calls semantic, thread-safety, portability on many UNIX-like systems, efficiency on a broad range of networks (up to giga-ethernet LAN). Moreover, this library is available free of charge under the LGPL[1] license at `http://www.loria.fr/~ejeannot/adoc`.

4.1. AdOC library API

The AdOC library Application Programming Interface is very small and provides the ability to send and receive arrays of data or files. It also provides the ability to force or disable compression. The 7 functions of the API are the following:

- `ssize_t adoc_send_file(int d, FILE *pf, ssize_t *slen)`. This function sends the file pointed by `pf` to the object referenced by the descriptor `d` (a socket for instance). After the call, the number of sent bytes is pointed by `slen`. The size of the file is returned by the function. The compression ratio is therefore the ratio between the value returned by the function and the value pointed by `slen`.
- `ssize_t adoc_send_file_levels(int d, FILE *pf, ssize_t *slen, unsigned int min, unsigned int max)`. This function is the same as above, except that `min` sets the minimum level of compression to be used and `max` sets the maximum level of compression to be used. Two internal constants `ADOC_MIN_LEVEL` and `ADOC_MAX_LEVEL` define the minimum and maximum values for `min` and `max`. For instance setting `max` to `ADOC_MIN_LEVEL`, disables the compression while setting `min` to `ADOC_MIN_LEVEL`+1, forces the compression.
- `ssize_t adoc_receive_file(int d, FILE *pf);`. It reads an AdOC stream from the object referenced by descriptor `d`, decompresses the data if necessary and stores the data into the file pointed by `pf`. The amount of data stored is returned by the function.
- `ssize_t adoc_write(int d, void *buf, size_t nbytes, ssize_t *slen)`. This function is the same as the `write` UNIX system call except that the number of sent bytes is output in the `slen` pointer (it can be set to `NULL` if not used by the application). It writes the data pointed by `buf` to the object referenced by the descriptor `d`. The maximum number of data to write is given by `nbytes`.The function returns `nbytes` on success (a negative value in case of failure). Thanks to compression, the number pointed by `slen` must be lower than `nbytes`.

1 `http://www.gnu.org/copyleft/lesser.html`

- `ssize_t adoc_write_levels(int d, void *buf, size_t nbytes, ssize_t *slen, unsigned int min, unsigned int max)`. This function is the same as above with the ability to force or disable compression.
- `ssize_t adoc_read(int d, void *buf,size_t nbytes);`. This function is the same as the `write` UNIX system call. It reads an AdOC stream from the object referenced by descriptor `d` and stores the uncompressed data into `buf`. The maximum number of bytes to read is given by `nbytes`. The actual number of bytes read is return by this function.
- `int adoc_close(int d)`. This function is used to close the descriptor file `d` and to free AdOC internal buffers. In order to respect the `read`/`write` system call semantic it is required to be able to perform partial read. For instance a sender can send 100 MB, and the receiver can perform two `read`s one of 60 MB and one of 40 MB. In this case, temporary buffers are allocated to store received data. If the socket is closed after a partial read, temporary buffers have to be freed.

The ability of AdOC to send files is provided to ease the use of the library when files are to be sent. It is not intended to be competitive to the `sendfile` system call provided by some UNIX systems (such as LINUX). The main reason is that the `sendfile` system call does the file copy inside the kernel whereas AdOC is a user level library. Only `adoc_read`, `adoc_write` and `adoc_close` are intended to be used instead of the corresponding system calls.

4.2. Thread safety

The library does not use any global variable. A static variable is used to store and retrieve internal buffers when performing partial read. This variable is always accessed between locks. Therefore, different threads can use AdOC at the same time[2].

4.3. Portability

This library has been ported and compiled on many UNIX-like systems. It incorporates the compression library required by the algorithm (zlib [10] and liblzf [13]). So far AdOC has been successfully compiled and tested on the following platform/architectures: Linux, Solaris/SunOS, Darwin/MacOS, freeBSD, IBM AIX, SGI IRIX, dec-alpha OSF, cygwin as well as 64 bits linux kernels. We also ported the AdOC library to gcc/windows. However, tests show that cygwin outperform the gcc/windows version in most of the cases. Therefore, due to the difficulty to maintain two versions we provide only the cygwin one.

Note that, since we use the liblzf and the zlib, the compression is lossless, and therefore no alteration of the data are seen by the user.

5. Performance issues

In Section 3, we described an overview of the AdOC algorithm. We discuss here some performance issues that we have dealt with. This requires to change the algorithm in order the library to be efficient in broad range of scenarii.

Fast Networks In order the AdOC library to be general, one should not see performance degradation on fast Network. For some networks (up to 100 Mbit LAN), we need fast compression libraries that are able to compress the data to a speed at least equal to that of the network. We use the LZF library of Marc Alexander Lehmann [13]. As shown in Table 1, it is a very fast compression library that has about the same speed as the `memcpy` function[3]. The drawback of this library is that the compression ratio is very low (less than 2) therefore, we use this library as the first compression level (the second compression level corresponds to gzip at level 1).

Furthermore, very fast networks such as Gbit LAN are too fast for modern processors to have time to compress data even with lzf. In order to avoid performance degradation for such networks, we incorporate a bandwidth measurement into the protocol as follow. If the size of the data to transmit is large enough (512 KB) we measure the time to transmit a part of the data (256 KB) without compression. We deduce the speed of the link. If this speed is above 500 Mb/s, it means that we are dealing with a very fast network and we send the remaining data uncompressed, otherwise we use the adaptive algorithm.

The drawback of this approach is that no compression is performed if the size of the data is less than 512 KB whatever the network is. We think that this is reasonable as we target mainly large data set transfers and that 512 KB is less than the half of a 3.25 inches floppy disk capacity.

Compression level divergence The goal of the AdOC algorithm is to maintain the emission queue size to a reasonable value. If the queue size is empty, this means that no packets are sent to the network. If it is too large, this means that we have time to compress the data. However, when the receiver is very slow with regard to the sender, the adaptation process may diverge. Indeed, if we start compressing the data, the receiver will take a longer time to decompress it.

2 We have incorporated AdOC into the Internet Backplane Protocol (IBP) [4] that use multiple threads to store or retrieve data from data handlers. It works without error.

3 We could have used lzo [14], which has comparable performance to lzf, but its license is incompatible with the AdOC one.

Usually the compression time is far longer than the decompression time because it requires more computation power, but this is no longer true when both ends are very heterogeneous. If the compression time becomes smaller than the decompression time and the network is fast enough, the queue size will increase leading to an increase of the compression level. This is not the good choice, because the receiver will still be the bottleneck, the queue size will increase again leading to an increase of the compression level, etc. The good choice would be to disable the compression in such case.

The problem is that we want the library to respect the read/write semantic. Therefore, it is not possible for the receiver to send any information to the sender and ask it to stop the compression. Hence, the sender has to guess that the receiver is too slow for the compressed data it is sending. In order to solve this problem, we propose the following conservative strategy. The compression thread continuously measure the visible bandwidth and records it for each compression level. When updating the compression level, AdOC checks if the current level gives a better visible bandwidth than any smaller compression levels. If this is not the case this means that maybe we are facing a compression level divergence (an other reason could be that the network is temporally congested). Nevertheless, our conservative strategy gets back to the level that gives a better visible bandwidth and forbids the previous compression level for 1 second. After 1 second, we assume that the dynamic condition may have changed and we let AdOC try this level again if it decides it can be useful.

With this strategy the compression level is disabled when the receiver is not able to decompress data at a rate greater than its network arrival speed.

Small messages The AdOC library is a multithreaded library with a queue that is shared between the threads and accessed by mutexes. This adds some latency to the transfer. This latency has a cost that is visible for short messages on fast networks. Nevertheless, for small messages, compression is not very useful, and we measure the speed of the network by sending the first 256 KB uncompressed. Therefore, when messages are short (less than 512 KB), the data are sent uncompressed directly without launching the threads. In this case, the latency is the same than direct read and write calls.

Compressed and random data Some data, like random or already compressed one, takes time to be compressed and the obtained compression ratio is poor and sometimes smaller than one. In AdOC sending such data can lead to performance degradation. In order to avoid such a degradation we compare the size of each compressed packet to its original size. If the compression ratio is smaller than a given threshold, we stop compressing the remaining of the buffer and set the compression level to its minimal value for the next 10 packets before enabling compression again.

6. Experiments

The AdOC Library is designed to be used as a general-purpose communication service for any application instead standard POSIX read/write system calls. Hence, we have first measured its performance against those calls. Second, as it is intended to be incorporated into grid middlewares, we have plugged AdOC into the NetSolve [6] and compared application performance of both versions.

6.1. AdOC vs. POSIX read/write

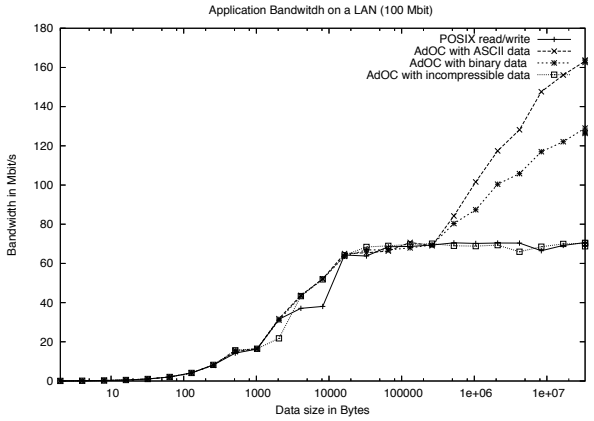


Figure 3. ***Bandwidth on a Fast Ethernet LAN***

6.1.1. Bandwidth Figures 3, 5, 4, 6 and 7 show the performance of AdOC compared to the POSIX read/write system calls. The experiments where performed using Linux machine, with 100 Mb network cards. On the x-axis, is shown the amount of transferred data in bytes. This axis use logarithmic scale. The sent data size is between 1 byte and 32 MB. On the y-axis, we show the bandwidth visible at the application level (by the user). It is evaluated by measuring the amount of time required by the application to send and received back a buffer of the given size.

Since the performance of AdOC depends on its capacity to compress data 4 drawings are shown on each figure. One represents the read/write performance. The three other drawings represent the AdOC timings with different data types. The first type represents ASCII data: it has a compression ratio of about 5 with gzip level 6. The second type represents binary data: it has a compression ratio of about 2 with gzip level 6. The last type represents incompressible

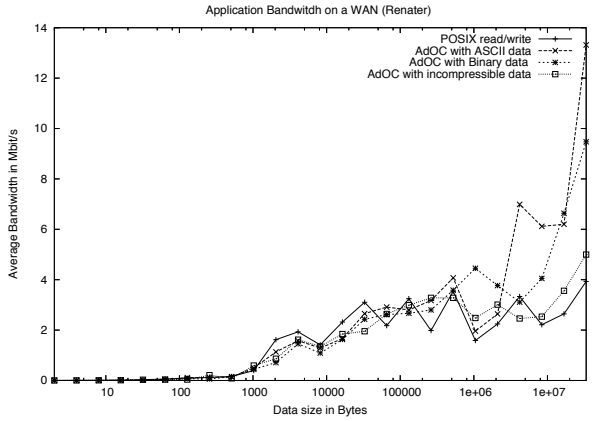


Figure 4. ***Bandwidth on Renater (Academic Network, between Nancy and Lyon), Average Timings***

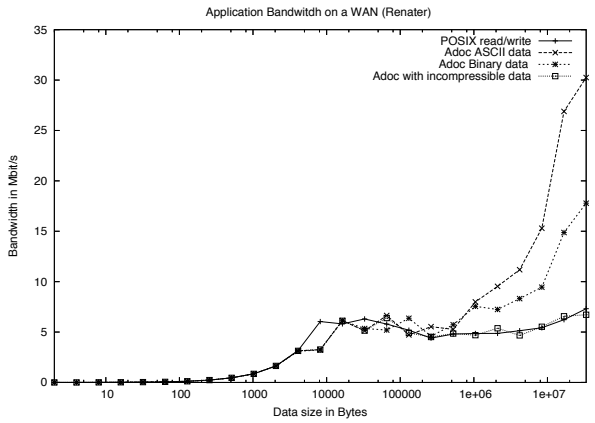


Figure 5. ***Bandwidth on Renater (Academic Network, between Nancy and Lyon), Best Timings***

data as gzip is not able to compress it. These data were generated randomly, the randomness being set accordingly to the desired compression ratio.

We believe that for most of the applications, data to be sent will be between the ASCII and the binary data.

Reproductivity of the experiments is a difficult issue. This is especially true on Internet and WAN where experiments are not reproducible. The standard deviation of the timings is very high and therefore it is difficult to conclude on the performance of each method based on average timings. To illustrate this phenomena, let us compare Figure 5 and Figure 4. On Fig. 4, each point represents the average

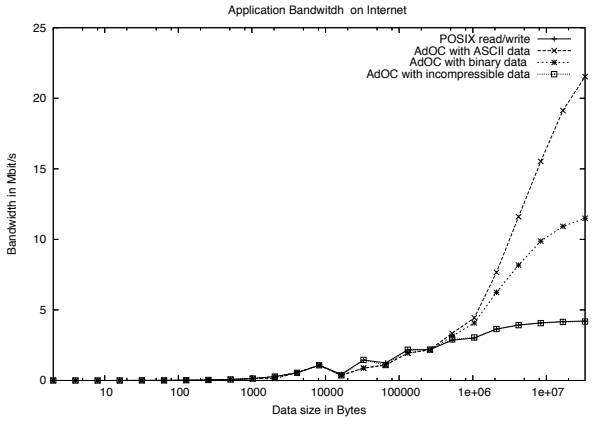


Figure 6. ***Bandwidth on Internet (Tennessee – France)***

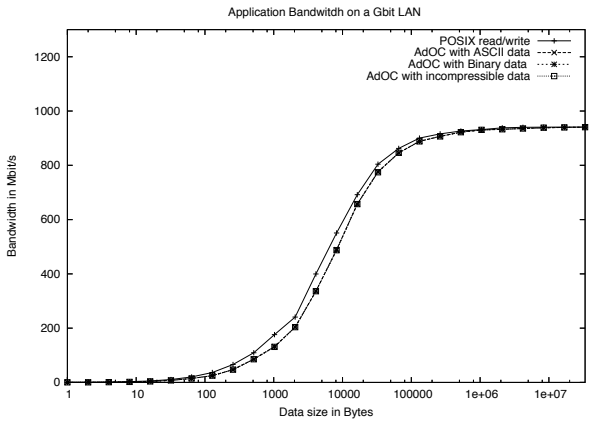


Figure 7. ***Bandwidth on a Gbit Ethernet LAN***

time of 40 measurements, on the Fig. 5, each point is the best time of 40 measurements. On one hand, we see that it is difficult to conclude on the behavior of each method with the plotting of average value (the average bandwidth is oscillating after 8 KB). On the other hand, plotting the best value gives smoother plots. Best-value plottings appear to be more reproducible. Therefore, we have decided to use only best values for *Renater* and *Internet* figures of this article. Another justification is that best value is fair for all the methods (with or without AdOC) : each of them is evaluated under the same circumstances: when network perturbation is minimal.

Results show that up to 512 KB, AdOC and POSIX read/write have the same performance: this is due to the fact that no compression is performed under this size. At that point and after, AdOC starts compressing data, and the time

	POSIX read/write	AdOC	AdOC with forced compression
Internet	80	80	225
Renater	9.2	9.2	25
100 Mbit LAN	0.18	0.20	1.8
Gbit LAN	0.030	0.045	1.6

Table 2. Latency of AdOC vs. POSIX read/write on Different Networks (in milliseconds)

to send data with AdOC becomes smaller than the time to send data with POSIX read/write. Not surprisingly the gain depends on the data and the network:

- On a 100 Mb Ethernet LAN (Fig. 3), for 32 MB, AdOC is between 1.85 and 2.36 times faster than the POSIX read/write.
- On Renater[4] between Nancy and Lyon (Fig. 4), for 32 MB, AdOC is between 6.1 and 2.6 times faster than the POSIX read/write.
- On Internet between France and Tennessee (Fig. 6), for 32 MB, AdOC is between 5.5 and 6 times faster than the POSIX read/write. The fact that the gain is smaller with Internet that with Renater is partially due to the fact that the machine we used in Tennessee was slower than the machines we used on Renater.

Moreover, we see that, for all these networks, for every size and type of data there is no performance degradation. Finally, the difference between AdOC with incompressible data and POSIX read/write is never significant: AdOC does not loose time with this kind of data.

For Gbit Ethernet (Fig. 7), we see a small degradation up to 1MB. This is the overhead of testing the network and the data size in order to guess if compression has to be used. However, in our tests the degradation does not depend on the size of the data: the overhead is between 10 and 20 μs.

6.1.2. Latency We have measured the average time of a zero byte ping-pong with AdOC and POSIX read/write. Results are shown in Table 2

We see that there is no difference between AdOC and POSIX read/write up to 100 Mb LAN. For Gbit LAN the latency is about 15 μs higher with AdOC. The latency given in the column AdOC with forced compression shows the overhead of starting the full AdOC process (threads, FIFO queue, mutexes, etc.) and the protocol overhead. Since these timings are very high, it justifies not to compress the data for small size.

4 Renater is the network that interconnects research center and university of france it provoides a backbone of several Gbit see `www.renater.fr`

6.2. AdOC into NetSolve

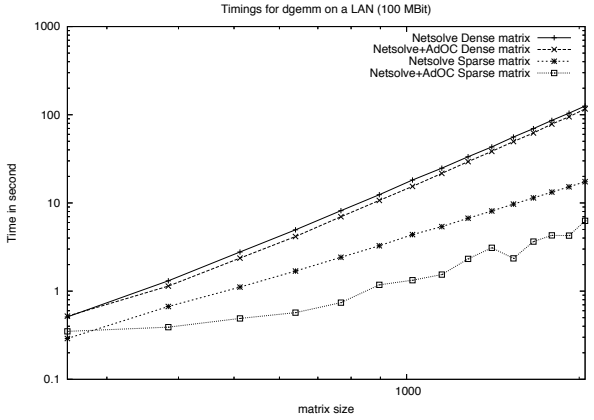


Figure 8. *NetSolve Timings on a 100 Mb LAN*

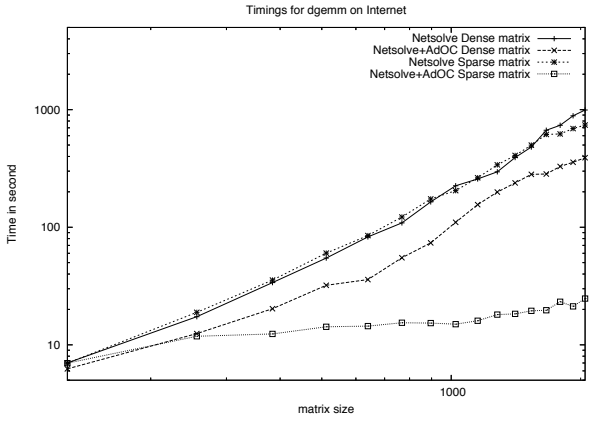


Figure 9. *NetSolve Timings on Internet (Tennessee-France)*

NetSolve is a middleware that works under the GridRPC model. It features a set of servers that register to an agent. When a client requests for a service it asks the agent to find the best suited server. It then executes the request to the server as a normal RPC.

We have modified NetSolve in order to enable AdOC in this middleware. This was very easy as it required to modify only the `communicator.c` file. We changed each

`read` call into `adoc_read` and each write call into `adoc_write`. We had also to change the makefiles so that NetSolve links against the AdOC library at compilation.

In Figures 8 and 9 we show the time to execute a dgemm[5] request on a LAN or on Internet using NetSolve. The agent and the server were on one end of the network whereas the client was on the other end. On the x-axis is shown the size of the matrix (number of lines or columns as matrices are square). On the y-axis we plot the time to perform the entire request. Each axis uses logarithmic scale. Two kinds of matrices where used. Matrix full of zero (called sparse matrix), matrix with 13 significant digits (as in some standard matrix libraries) and an exponent between 10^{-20} and 10^{+20} (called dense matrix). We do not use `oilpann.hb` file as it is a fix size ASCII matrix and we want to vary the size and use binary data. In our case a sparse matrix is very easy to compress : it is the best case. A dense matrix is hard to compress and should be considered as the worst realistic case.

For each kind of data the time with and without AdOC is plotted.

On a LAN (Fig. 8), we see that for dense matrices NetSolve with AdOC is slightly better than NetSolve without AdOC (5% faster for 2048*2048 matrix). On sparse matrix performance is better (up to 5.6 times faster for a 2048*2048 matrix). There is no performance degradation due to AdOC for any matrix size and any data type.

On Internet (Fig. 9), we see that NetSolve with AdOC always outperforms standard NetSolve. It is 2.6 times faster on a 2048*2048 dense matrix and 30.8 times faster on a 2048*2048 sparse matrix. We never see performance degradation due to AdOC on Internet too.

7. Related work

Several researches are done on using compression for transmitting data. In [12], the authors proposed an algorithm closed to the AdOC algorithm. They implemented this algorithm in the linux Kernel (TCP stack). Hence this implementation was not portable. With these authors we proposed the AdOC algorithm in [11].

In [18] the authors proposed a work close to ours. The adaptivity depends on the network, CPU and data. However, it ignores any related work on adaptive compression and this work is less general than ours as no library is provided and it does not work on 100 Mb LAN or higher. For high speed compression, it uses the Huffman algorithm that is slower and gives lower compression ratio than LZF.

In [19], an other adaptive compression study is performed. This is an ongoing work. This work highlights some problems of the original AdOC algorithm. These problems are all addressed in this paper. The compression is performed using threaded and non-threaded implementation. In the non-threaded implementation there is no overlap of communication and compression. It proposed a feedback mechanism in order to avoid compression level divergence. However, this mechanism requires to know the maximum available bandwidth of the network.

Compression to speedup data transfer is used in [2]. In this work the authors propose a Grid-enable computational framework based on Cactus [3] and Globus [9]. However, the compression was not adaptive: once, the compression is set, it is not possible to disabled it.

In [7], the authors propose an integrated solution for wide area communication on grids called NetIbis. Many features are proposed in this work and they use AdOC for enhancing the communication performance.

8. Conclusion

Data transfer is a key feature for computational and data grids. Such grids have to rely on efficient data transmission services that are able to provide fast transfer rate. Compression is one mean to increase the bandwidth see at the application level. However, the heterogeneous and dynamic nature of the grids required to adapt the compression to the environment.

In this paper we have presented the AdOC library. This library provides adaptive online compression for transferring data. The main features of the AdOC library are:

- The compression level is adapted according to the environment (current speed of the network and CPUs) and the data. The compression is lossless.
- It provides compression and communication overlap. AdOC is able to compress some part of the data while sending compressed or uncompressed packets.
- It works on a broad range of network (up to Gbit LAN)
- It is easy to incorporate into any existing software. AdOC is thread-safe and its API is very close to the read/write system calls and respects their semantics.
- It is ported on many UNIX like systems (LINUX (32/64 bits), SunOS, Darwin, Cygwin, etc.)
- It has a low latency: for small messages AdOC gives the same performance as POSIX read/write (up to 100 MBit LAN).

We have tested this library on various condition with various data types. First, it appears that there is almost no performance degradation due to AdOC (on Giga-Ehternet LAN, some microseconds are lost due to AdOC). Second, the performance gain obtained using AdOC depends on the data itself and the environment and can be very important (up to 6 times faster).

5 A dgemm is a matrix-multiplication program.

This library is intended to be used in any middleware that performs data transfer. We have incorporated the library into NetSolve. This was done easily thanks to the API close to the read and write system call. The performance of NetSolve with AdOC is never worst than NetSolve alone. Most of the results show an increase of performance for NetSolve with AdOC.

Our future work is directed towards extending the use of AdOC in existing software. An IBP data mover has already been proposed: it is needed to evaluate the performance precisely. The next software we target is gridFTP [1], where (as in FTP) a compression option is available.

We also direct our future work towards lossy compression for image transfer with various resolution. This is useful when a user has to choose one image among a set of images (thumbnails): the resolution and accuracy of the thumbnails is not necessary required to be very high.

9. Acknowledgments

We would like to thanks several people involved in the development of this library.

Aurelien Bouteiller from the LRI beta tested AdOC and fixed some bugs.

Ndoli-Guillaume Assielou, Bertrand Benoit and Alexandre Dombrat tested fast compression libraries such as LZF.

Hanane Moustain Billah worked on porting AdOC on the windows system (both natively and with Cygwin).

This work is partially supported by the french ACI GRID AGIR of the Ministry of Research.

References

[1] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, and S. Tuecke. *GridFTP Protocol Specification*. GGF GridFTP Working Group Document, 2002.

[2] G. Allen, T. Dramlitsch, I. Foster, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus. In *Conference on Supercomputing (SC 2001)*, Dever, Colorado, Nov. 2001.

[3] G. Allen, T. Goodale, and E. Seidel. The Cactus computational Collaboratory: Enabling Technologies for Relativistic Astrophysics, and a Toolkit for solving PDEs by Communities in science and engineering. In *7th IEEE Symposium on the Frontiers of Massively Parallel Computation (Frontiers'99)*, New-York, USA, 1999.

[4] A. Bassi, M. Beck, T. Moore, J. S. Plank, M. Swany, R. Wolski, and G. Fagg. The Internet Backplane Protocol: A Study in Resource Sharing. *Future Generation Computing Systems*, 19(4):551–561, May 2003.

[5] E. Caron, F. Desprez, F. Lombard, J.-M. Nicod, M. Quinson, and F. Suter. A Scalable Approach to Network Enabled Servers. In B. Monien and R. Feldmann, editors, *Proceedings of the 8th International EuroPar Conference*, volume 2400 of *Lecture Notes in Computer Science*, pages 907–910, Paderborn, Germany, Aug. 2002. Springer-Verlag.

[6] H. Casanova and J. Dongarra. NetSolve : A Network Server for Solving Computational Science Problems. In *Proceedings of Super-Computing'96*, Pittsburg, 1996.

[7] A. Denis, O. Aumage, R. Hofman, K. Verstoep, T. Kielmann, and H. E. Bal. Wide-Area Communication for Grids: An Integrated Solution to Connectivity, Performance and Security Problems. In *Proceedings. of the 13th IEEE International Symposium on High Performance Distributed Computing*, pages 97 – 106, June 2004.

[8] P. Deutsch. *GZIP file format specification version 4.3*. IETF, Network Working Group, 1996. RFC1952.

[9] I. Foster and C. Kesselman. The Globus project: A progress report. In *Heterogeneous Computing Workshop*, Mar. 1998.

[10] J.-l. Gailly and M. Adler. zlib. `http://www.gzip.org/zlib/`.

[11] E. Jeannot, B. Knutsson, and M. Björkman. Adaptive Online Data Compression. In *IEEE High Performance Distributed Computing (HPDC'11)*, pages 379 – 388, Edinburgh, Scotland, July 2002.

[12] B. Knutsson and M. Björkman. Trading computation for communication by end-to-end compression. In *Third International Workshop on High Performance Protocol Architectures (HIPPARCH'97)*, 1997.

[13] M. A. Lehmann. liblzf. `http://www.goof.com/pcg/marc/liblzf.html`.

[14] M. F. Oberhumer. Lzo. `http://www.oberhumer.com/opensource/lzo/`.

[15] J. Postel and J. Reynolds. *FILE Transfer Protocol (FTP)*. IETF, Network Working Group, 1985. RFC 959.

[16] D. Rand. *The PPP Compression Control Protocol (CCP)*. IETF, Network Working Group, 1996. RFC1962.

[17] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H.Takagi. Ninf: A Network based Information Library for a Global World-Wide Computing Infrastracture. In *HPCN'97 (LNCS-1225)*, pages 491–502, 1997.

[18] K. Schwan, P. Widener, and Y. Wiseman. Efficient End to End Data Exchange Using Configurable Compression. In *24th International Conference on Distributed Computing System (ICDCS 2004)*, Tokyo, Japan, Mar. 2004.

[19] L. Singaravelu, L. Kang, S. Park, and C. Pu. Effectiveness of Data Compression in Networks. Draft at `www.cc.gatech.edu/grads/l/lenin/docs/AC.ps`.

[20] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, May 1977.

[21] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

Managing Data Persistence in Network Enabled Servers*

Eddy Caron
GRAAL Project
LIP ENS Lyon, 46 Allée d'Italie
69364 Lyon Cedex 07, France
Eddy.Caron@ens-lyon.fr

Bruno DelFabbro
GRAAL Project
LIFC, Université de Franche-Comté
16 route de Gray, 25030 Besançon Cedex, France
delfabbro@lifc.univ-fcomte.fr

Frédéric Desprez
GRAAL Project
LIP ENS Lyon, 46 Allée d'Italie
69364 Lyon Cedex 07, France
Frederic.Desprez@ens-lyon.fr

Emmanuel Jeannot
ALGORILLE Project
LORIA†INRIA-Lorraine
Nancy, France
Emmanuel.Jeannot@loria.fr

Jean-Marc Nicod
GRAAL Project
LIFC, Université de Franche-Comté
16 route de Gray, 25030 Besançon Cedex, France
nicod@lifc.univ-fcomte.fr

April 3, 2006

Abstract

The GridRPC model [17] is an emerging standard promoted by the Global Grid Forum (GGF)[1] that defines how to perform remote client-server computations on a distributed architecture. In this model data are sent back to the client at the end of every computation. This implies unnecessary communications when computed data are needed by an other server in further computations. Since, communication time is sometimes the dominant cost of remote computations, this cost has to be lowered. Several tools instantiate the GridRPC model such as NetSolve developed at the University of Tennessee, Knoxville, USA, and DIET developed at LIP laboratory, ENS Lyon, France. They are usually called Network Enabled Servers (NES). In this paper, we present a discussion of the data management solutions chosen for these two NES (NetSolve and DIET) as well as experimental results.

1 Introduction

Due to the progress in networking, computing intensive problems from several areas can now be solved using network scientific computing. In the same way that the World Wide Web has changed

*This work was supported in part by the ACI GRID (ASP) and the RNTL (GASP) from the French ministry of research.

†UMR 7503, CNRS, INRIA, UHP Nancy-1, Univ. Nancy 2, INPL

[1] http://www.ggf.org

the way that we think about information, we can easily imagine the kind of applications we might construct if we had instantaneous access to a supercomputer from our desktop. The GridRPC approach [20] is a good candidate to build Problem Solving Environments on computational Grid. It defines an API and a model to perform remote computation on servers. In such a paradigm, a client can submit a request for solving a problem to an agent that chooses the best server amongst a set of candidates. The choice is made from static and dynamic information about software and hardware resources. Request can be then processed by sequential or parallel servers. This paradigm is close to the RPC (*Remote Procedure Call*) model. The GridRPC API is the Grid form of the classical Unix RPC approach. They are commonly called Network Enabled Server (NES) environments [16].

Several tools exist that provide this functionality like NetSolve [7], Ninf [13], DIET [4], NEOS [18], or RCS [1]. However, none of them do implement a general approach for data persistence and data redistribution between servers. This means that once a server has finished its computation, output data are immediately sent back to the client and input data are destroyed. Hence, if one of these data is needed for another computation, the client has to bring it back again on the server. This problem as been partially tackled in NetSolve with the request sequencing feature [2]. However, the current request sequencing implementation does not handle multiple servers.

In this paper, we present how data persistence can be handled in NES environments. We take two existing environments (NetSolve and DIET) and describe how we implemented data management in their kernels. For NetSolve, it requires to change the internal protocol, the client API and the request scheduling algorithm. For DIET we introduce a new service, called the Data Tree Manager (DTM), that identify and manage data within this middleware. We evaluate the gain that can be obtained from these features on a grid. Since we show that data management can greatly improve application performance we discuss a standardization proposal.

The remaining of this paper is organized as follows. In Section 2, we give an overview of Network Enabled Server (NES) architecture. We focus on NetSolve and DIET. We show why this is important to enable data persistence and redistribution to NES. We describe how we implemented data management in NetSolve and DIET respectively in Section 3 and in Section 4. Experimental results are presented in Section 5. In Section 6 we discuss the standardization of data management in NES. Finally, Section 7 concludes the paper.

2 Background

2.1 Network Enabled Server Architectures

2.1.1 General Architecture

The NES model defines an architecture for executing computation on remote servers. This architecture is composed of three components:

- the *agent* is the manager of the architecture. It knows the state of the system. Its main role is to find servers that will be able to solve as efficiently as possible client requests,
- *servers* are computational resources. Each server registers to an agent and then waits for client requests. Computational capabilities of a server are known as problems (matrix multi-

plication, sort, linear systems solving, etc.). A server can be sequential (executing sequential routines) or parallel (executing operations in parallel on several nodes),

- a *client* is a program that requests for computational resources. It asks the agent to find a set of servers that will be able to solve its problem. Data transmitted between a client and a server is called object. Thus, an input object is a parameter of a problem and an output object is a result of a problem.

The NES architecture works as follows. First, an agent is launched. Then, servers register to the agent by sending information of problems they are able to solve as well as information of the machine on which they are running and the network's speed (latency and bandwidth) between the server and the agent. A client asks the agent to solve a problem. The agent scheduler selects a set of servers that are able to solve this problem and sends back the list to the client. The client sends the input objects to one of the servers. The server performs the computation and returns the output objects to the client. Finally local server objects are destroyed.

This client API for such an approach has been standardized within the Global Grid Forum. The GridRPC working group [12] proposed an API that is instantiated by several middleware such as DIET, Ninf, NetSolve, and XtremWeb.

2.1.2 NetSolve

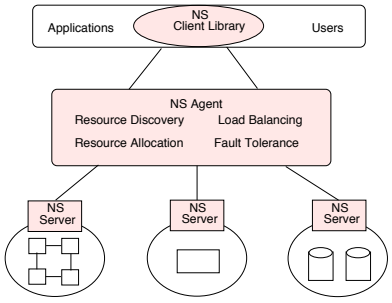


Figure 1: NetSolve Architecture.

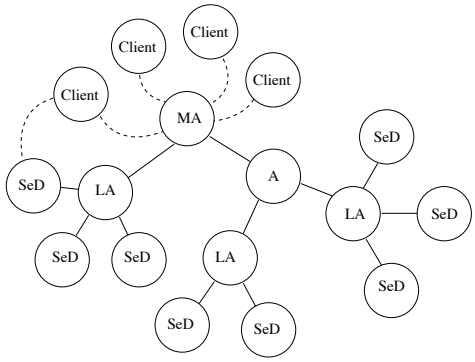


Figure 2: DIET Architecture.

NetSolve [7] (Figure 1) is a tool built at the University of Tennessee and instantiate the GridRPC model described above. It is out of the scope of this paper to completely describe NetSolve in detail. In this section we focus only on data management.

Request Sequencing. In order to tackle the problem of sending to much data on the Network, the *request sequencing* feature has been proposed since NetSolve 1.3 [2]. Request sequencing consists in scheduling a sequence of NetSolve calls on one server. This is a high level functionality since only two new sequence delimiters `netsl_sequence_begin` and `netsl_sequence_start` are added in the client API. The calls between those delimiters are evaluated at the same time and the data movements due to dependencies are optimized.

However request sequencing has the following deficiencies. First, it does not handle multiple servers because no redistribution is possible between servers. An overhead is added to schedule

NetSolve requests. Indeed, the whole Directed Acyclic Graph of all the NetSolve calls within the sequence is built before being sent to the chosen computational server. Second, `for` loops are forbidden within sequences, and finally the execution graph must be static and cannot depend on results computed within the sequence.

Data redistribution is not implemented in the NetSolve's request sequencing feature. This can lead to sub-optimal utilization of the computational resources when, within a sequence, two or more problems can be solved in parallel on two different servers. This is the case, for instance, if the request is composed of the problems *foo1, foo2 and foo3* given Figure 4. The performance can be increased if *foo1* and *foo2* can be executed in parallel on two different servers.

Distributed storage Infrastructure. To make a data persistent and to take advantage of its placement in the infrastructure, NetSolve proposes the Distributed Storage Infrastructure. The DSI helps the user for controlling the placement of data that will be accessed by a server (see Figure 3). Instead of multiple transmissions of the same data, DSI allows the transfer of the data once from the client to a storage server. Considering these storage servers closer from computational servers than from the client, the cost of transferring data will be cheaper. NetSolve is able to manage several DSI. Currently, NetSolve proposes this storage service using IBP (Internet Backplane Protocol)[2]. Files or items managed by a DSI are called DSI objects. To generate a DSI data, the client has to know the server in which it wants to store its data. Note that the data location is not a criteria for the choice of a computational server. NetSolve maintains its own File Allocation Table to manage DSI objects. Typically, when a request is submitted to a NetSolve Server, the server looks for input data and verify its existence in its FAT. If the data is referenced (the client had passed a DSI object), data is get from the storage server, the server gets it from the client elsewhere.

DSI improves the data transfer but does not prevent from data going back and forth from computational servers to storage servers. Indeed, this feature does not fully implement data persistence and therefore may lead to over-utilization of the network.

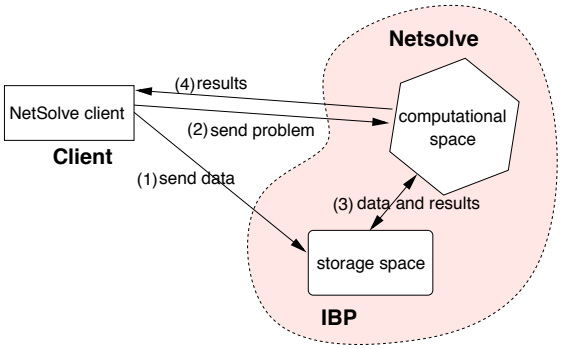


Figure 3: Distributed Storage Infrastructure.

2.1.3 DIET Architecture

NetSolve and Ninf projects are built on the same approach. Unfortunately, in these environments, it is possible to launch only one agent responsible of the scheduling for a given group of computational

[2] http://loci.cs.utk.edu/

servers[3]. The drawback of the mono-agent approach is that the agent can become bottleneck if a large number of requests have to be processed at the same time. Hence, NetSolve or Ninf cannot be deployed for large groups of servers or clients.

In order to solve this problem, DIET proposes to distributed the load of the agent work. It is replaced by several agents which organization follows two approaches: a peer-to-peer multi-agents approach that helps system robustness [6] and a hierarchical approach that helps scheduling efficiency [9]. This repartition offers two main advantages: first, we assume a better load balancing between the agents and a higher system stability (if one of the agents dies, a reorganization of the others is possible to replace it). Then, it is easier to manage each group of servers and agents by delegation which is useful for scalability. DIET is built upon several components:

- a *client* is an application that uses DIET to solve problems. Several client types must be able to connect to DIET. A problem can be submitted from a Web page, a problem solving environment such as Scilab [3] or Matlab or from a compiled program.

- a *Master Agent (MA)* is directly linked to the clients. It is the entry point of our environment and thus receives computation requests from clients attached to it. These requests refer to some DIET problems that can be solved by registered servers. Then the MA collects computation abilities from the servers and chooses the best one. A MA has the same information than a LA, but it has a global and high level view of all the problems that can be solved and of all the data that are distributed in all its subtrees.

- a *Leader Agent (LA)* forms a hierarchical level in DIET. It may be the link between a Master Agent and a SeD or between two Leader Agents or between a Leader Agent and a SeD. It aims at transmitting requests and information between Agents and several servers. It maintains a list of current requests and the number of servers that can solve a given problem and information about the data distributed in its subtrees.

- a *Server Daemon (SeD)* is the entry point of a computational resource. The information stored on an SeD is a list of the data available on its server (with their distribution and the way to access them), the list of problems that can be solved on it, and all information concerning its load (memory available, number of resources available, ...). A SeD declares the problems it can solve to its parent. For instance, a SeD can be located on the entry point of a parallel computer.

A new DIET client contacts a Master Agent (the closest for instance) and posts its request. The Master Agent transmits the request to its subtrees[4] to find data already present in the platform and servers that are able to solve the problem. The LAs which receive the request forward it down to every one of their sub-trees which contains a server that might be involved in the computation and wait for the responses. The requests traverse the entire hierarchy down to the SeDs. When a SeD receives a request, it sends a response structure to its father. It fills the fields for the variables it owns, leaving a null value for the others. If it can solve the problem, it also puts an entry with

[3]In Ninf, a multi-agents platform exists (Metaserver) but each agent has the global knowledge of the entire platform.

[4]an extension is possible for the multi-agent approach: broadcast the request to the others MA considering them as Leader Agents

its evaluated computation time acquired from our performance forecasting tool FAST [19]. Each LA gathers responses coming from its children and aggregates them into a structure.

The scheduling operations are realized at each level of the tree when the response is sent back to the Master Agent. Note that a time-out is set and when an agent has not got a response over a given time, this response is ignored. However, this time-out is not an information enough to say that an agent has failed. When the responses come back to the MA, it is able to take a scheduling decision. The evaluated computation and communication times are used to find the server with the lowest response time to perform the computation. Then the MA is able to send the chosen server reference to the client (it is also possible to send a bounded list of best servers to the client). Then, the Master Agent orders the data transfer. Here we can distinguish two cases: data resides in the client and are transferred from the client to the chosen server or data are already inside the platform and are transferred from the servers that holds them to the chosen server. Note that these two operations can be processed in a parallel way. Once data are received by the server, computation can be done. The results may be sent to the client. For performance issues, data are let in the last computational server if possible.

2.2 On the Importance of Data Management in NES

A GridRPC environment such as NetSolve and DIET is based on the client-server programming paradigm. This paradigm is different than other ones such as parallel/distributed programming. In a parallel program (written in PVM or MPI for instance) data persistence is performed implicitly: once a node has received some data, this data is supposed to be available on this node as long as the application is running (unless explicitly deleted). Therefore, in a parallel program, data can be used for several steps of the parallel algorithm.

However, in a GridRPC architecture no data management is performed. Like in the standard RPC model, request parameters are sent back and forth between the client and the server. A data is not supposed to be available on a server that used it for another step of the algorithm (an new RPC) once a step is finished (a previous RPC has returned). This drawback can lead to very high execution time as the execution and the communications can be performed over the Internet.

2.2.1 Motivating Example

Now we give an example where the use of data persistence and redistribution improves the execution of a GridRPC session. Assume that a client asks to execute the three functions/problems shown in the sample code given in Figure 4(a).

Let us consider that the underlying network between the client and the server has a bandwidth of 100 Mbit/s (12.5 Mbytes per seconds). Figure 4(b) gives the execution time for each function and for each server. Finally let us suppose that each object has a size of 25 Mbytes. The GridRPC architecture will execute *foo1* and *foo3* on server S_1 and *foo2* on S_2 and sends the objects in the following order: b, c, e, f (Figure 4). Due to the bandwidth limitation, *foo1* will start 4 seconds after the request and *foo2* after 8 seconds. Without data persistence and redistribution a will be available on S_1 16 seconds after the beginning of the session and d, 18 seconds after the beginning (S_2 has to wait that the client has completely received a before starting to send d). Therefore, after the execution of *foo3*, g will be available on the client 26 seconds after the beginning. With data persistence and redistribution, S_2 sends d to S_1 which is available 13 seconds after the beginning of

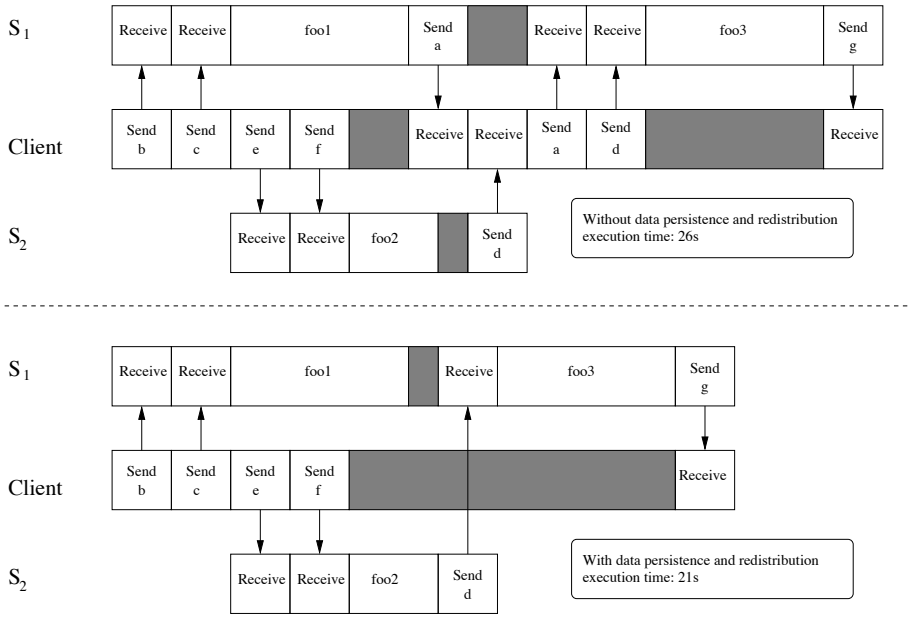
the request. Hence, g will be available on the client 21 seconds after the beginning of the request which leads to a 19% improvement.

```
a = foo1(b,c)
d = foo2(e,f)
g = foo3(a,d)
```

(a) Sample C code.

Function	Server 1	Server 2
foo1	6s	9s
foo2	2s	3s
foo3	6s	11s

(b) Execution time.



(c) Execution without (top) and with (bottom) persistence.

Figure 4: Sample example where data persistence and redistribution is better than retrieving data to the client.

2.2.2 Goal of the Work

In this paper, we show how to add data management into NES environments. We added data persistence and data redistribution to NetSolve and DIET and therefore modified the client API.

Data persistence consists in allowing servers to keep objects in place to be able to use these objects again for a new call without sending them back and forth from and to the client. Data redistribution enables inter-server communications to avoid object moving though the client.

Our modifications to NetSolve are backward compatible. Data persistence and data redistribution require the client API to be modified but standard client programs continue to execute normally. Moreover, our modifications are stand-alone. This means that we do not use an other software to implement our optimizations. Hence, NetSolve users do not have to download and

compile new tools. Finally, our implementation is very flexible without the restrictions imposed by NetSolve's request sequencing feature.

We also proposed a model of distributed data management in DIET. The DIET data management model is based on two key elements: the data identifiers and the Data Tree Manager (DTM) [11, 10]. To avoid multiple transmissions of the same data from a client to a server, the DTM allows to leave data inside the platform after computation while data identifiers will be used further by the client to reference its data.

3 New Data Management in NetSolve

In this section we describe how we have implemented data redistribution and persistence within NetSolve. This required to change the three components of the software: server, client, and agent.

3.1 Server Modifications

NetSolve communications are implemented using sockets. In this section, we give details about the low level protocols that enable data persistence and data redistribution between servers.

3.1.1 Data Persistence

When a server has finished its computation, it keeps all the objects locally, listen to a socket and waits for new orders from the client. So far, the server can receive five different orders.

1. *Exit.* When this order is received, the server terminates the transaction with the client, exits, and therefore data are lost. Saying that the server exits is not completely correct. Indeed, when a problem is solved by a server, a process is forked, and the computation are performed by the forked process. Data persistence is also done by the forked process. In the following, when we say that the server is terminated, it means that the forked process exits. The NetSolve server is still running and it can solve new problems.

2. *Send one input object.* The server must send an input object to the client or to an other server. Once this order is executed, data are not lost and the server is waiting for new orders.

3. *Send one output object.* This order works the same way than the previous one but a result is sent.

4. *Send all input objects.* It is the same as "send one input object" but all the input objects are sent.

5. *Send all output objects.* It is the same as "send one output object" but all the results are sent.

3.1.2 Data Redistribution

When a server has to solve a new problem, it has first to receive a set of input objects. These objects can be received from the client or from an other server. Before an input object is received, the client tells the server if this object will come from a server or from the client. If the object comes from the client, the server has just to receive the object. However, if the object comes from

an other server, a new protocol is needed. Let call S_1 the server that has to send the data, S_2 the server that is waiting for the data, C and the client.

1. S_2 opens a socket s on an available port p.
2. S_2 sends this port to C.
3. S_2 waits for the object on socket s.
4. C orders S_1 to send one object (input or output). It sends the object number, forward the number of the port p to S_1 and sends the hostname of S_2.
5. S_1 connects to the socket s on port p of S_2.
6. S_1 sends the object directly to S_2 on this socket: data do not go through the client.

3.2 Client Modifications

3.2.1 New Structure for the Client API

When a client needs a data to stay on a server, three informations are needed to identify this data. (1) Is this an input or an output object? (2) On which server can it be currently found? (3) What is the number of this object on the server? We have implemented the `ObjectLocation` structure to describe these informations needed. `ObjectLocation` has 3 fields:

1. `request_id` which is the request number of the non-blocking call that involves the data requested. The request id is returned by the `netslnb` standard NetSolve function, that performs a non blocking remote execution of a problem. If `request_id` equals -1, this means that the data is available on the client.
2. `type` can have two values: `INPUT_OBJECT` or `OUTPUT_OBJECT`. It describes if the requested object is an input object or a result.
3. `object_number` is the number of the object as described in the problem descriptor.

3.2.2 Modification of the NetSolve Code

When a client asks for a problem to be solved, an array of `ObjectLocation` data structures is tested. If this array is not `NULL`, this means that some data redistribution have to be issued. Each element of the array corresponds to an input object. For each input object of the problem, we check the `request_id` field. If it is smaller than 0, no redistribution is issued, everything works like in the standard version of NetSolve. If the `request_id` field is greater than or equal to zero then data redistribution is issued between the server corresponding to this request (it must have the data), and the server that has to solve the new problem.

3.2.3 Set of New Functions

In this section, we present the modifications of the client API that uses the low-level server protocol modifications described above. These new features are backward compatible with the old version. This means that an old NetSolve client will have the same behavior with this enhanced version: all the old functions have the same semantic, except that when starting a non-blocking call, data stay on the server until a command that terminates the server is issued. These functions have been implemented for both C and Fortran clients. They are very general and can handle various situations. Hence, unlike request sequencing, no restriction is imposed to the input program. In Section 3.4, a code example is given that uses a subset of these functions.

Wait Functions. We have modified or implemented three functions: `netslwt`, `netslwtcnt` and `netslwtnr`. These functions block until the current computations are finished. With `netslwt`, the data are retrieved and the server exits. With `netslwtcnt` and `netslwtnr`, the server does not terminate and other data redistribution orders can be issued. The difference between these two functions is that unlike `netslwtcnt`, `netslwtnr` does not retrieve the data.

Terminating a Server. The `netslterm` orders the server to exit. The server must have finished its computation. Local object are then lost.

Probing Servers. As in the standard NetSolve, `netslpr` probes the server. If the server has finished its computation, results are not retrieved and data redistribution orders can be issued.

Retrieving Data. A data can be retrieved with the `netslretrieve` function. Parameters of this functions are the type of the object (input or output), the request, the object number and a pointer where to store the data.

Redistribution Function. `netslnbdist`, is the function that performs the data redistribution. It works like the standard non-blocking call `netslnb` with one more parameter: an `ObjectLocation` array, that describes which objects are redistributed and where they can be found.

3.3 Agent Scheduler Modifications

The scheduling algorithm used by NetSolve is Minimum Completion Time (MCT) [15] which is described in Figure 5. Each time a client sends a request MCT chooses the server that minimizes the execution time of the request assuming no major change in the system state.

We have modified the agent's scheduler to take into account the new data persistence features. The standard scheduler assumes that all data are located on the client. Hence, communication costs do not depend on the fact that a data can already be distributed. We have modified the agent's scheduler and the protocol between the agent and the client in the following way. When a client asks the agent for a server, it also sends the location of the data. Hence, when the agent computes the communication cost of a request for a given server, this cost can be reduced by the fraction of data already hold by the server.

1 **For all** server S that can resolve the problem
2 $D_1(S)$ = estimated amount of time to transfer input and output data.
3 $D_2(S)$ = estimated amount of time to solve the problem.
4 Choose the server that minimizes $D_1(S) + D_2(S)$.

Figure 5: MCT algorithm.

3.4 Code Example

In Figure 6 we show a code that illustrates the features described in this paper. It executes 3 matrix multiplications: `c=a*b`, `d=e*f`, and `g=d*a` using the `DGEMM` function of the level 3 BLAS provided by NetSolve, where `a` is redistributed from the first server and `d` is redistributed from the second one. We will suppose that matrices are correctly initialized and allocated. In order to simplify this example we will also suppose that each matrix has `n` rows and columns and tests of requests are not shown.

In the two `netslnb` calls different parameters of `dgemm` ($c = \beta \times c + \alpha \times a \times b$, for the first call) are passed such as the matrix dimension (always `n` here), the need to transpose input matrices (not used here), the value of α and β (respectively 1 and 0) and pointers to input and output objects. All these objects are persistent and therefore stay on the server: they do not move back to the client.

Then the redistribution is computed. An array of `ObjectLocation` is build and filled for the two objects that need to be redistributed (`a` and `d`). The call to `netslnbdist` is similar to previous `netslnb` call except that the redistribution parameter is passed. At the end of the computation, a *wait* call is performed for the computation of `g`, the matrix `c` is retrieved and the server that computed `d` is terminated.

In Section 5.2, we present our experimental results on executing a set of `DGEMM` requests both on a LAN and on a WAN.

4 Data Management in DIET

We have developed a data management service in the DIET platform. Our motivation was based on the need to decrease the global computation time. A way to achieve such a goal is to decrease data transfers between clients and the platform when possible. For example, a client that submits two successive calls with the same input data needs to transfer them twice (see Figure 7). Our goal is to provide a service that allows only one data transfer as shown in Figure 8. An other objective is to allows the use of the data already stored inside the platform in later computations and more generally in later sessions or by others clients. This is why data stored needed to be handled by an unique identifier. Our service has also to fit with DIET platform characteristics, and this is why our components are build in a hierarchical way. After a short description of the principles we retain in order to build a data management service in DIET, we review the various components of

```
ObjectLocation *redist;

netslmajor("Row");
trans="N";
alpha=1;
beta=0;

/* c=a*b */
request_c=netslnb("DGEMM()",&trans,&trans,n,n,n,&alpha,a,n,b,n,&beta,c,n);
/* after this call c is only on the server */

/* d=e*f */
request_d=netslnb("DGEMM()",&trans,&trans,n,n,n,&alpha,e,n,f,n,&beta,d,n);
/* after this call d is only on the server */

/* COMPUTING REDISTRIBUTION */
/* 7 input objects for DGEMM */
nb_objects=7;
redist=(ObjectLocation*)malloc(nb_objects*sizeof(ObjectLocation));

/* All objects are first supposed to be hosted on the client */
for(i=0;i<nb_object;i++)
  redist[i].request_id=-1;

/* We want to compute g=d*a */

/* a is the input object No 4 of DGEMM and the input object No 3 of request_c */
redist[4].request_id=request_c;
redist[4].type=INPUT_OBJECT;
redist[4].object_number=3;

/* d is the input object No 3 of DGEMM and the output object No 0 of request_d */
redist[3].request_id=request_d;
redist[3].type=OUTPUT_OBJECT;
redist[3].object_number=0;

/* g=d*a */
request_g=netslnbdist("DGEMM()",redist,&trans,&trans,n,n,n,&alpha,NULL,n,NULL,n,
                      &beta,g,n);

/* Wait for g to be computed and retrieve it */
netslwt(request_g);

/* retrieve c */
netslretrieve(request_c,OUTPUT_OBJECT,0,c);

/* Terminate the server that computed d */
netslterm(request_d);
```

Figure 6: NetSolve persistence code example.

our implementation called Data Tree Manager [10].

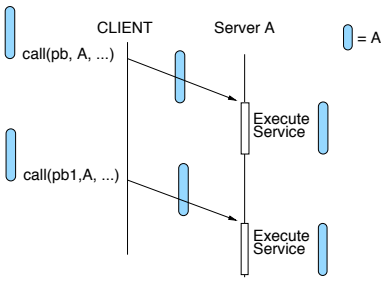


Figure 7: Sending A twice

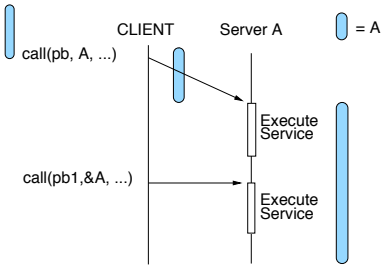


Figure 8: Sending A only once

Figure 9: Two successive calls

4.1 Principles

In this section, we present the basic functionalities that we choose for a data management service in such an ASP environment.

Data Storage A data can be stored onto a disk or in memory. In NES environments, a challenge is to store data as near as possible to a computational server where they will be needed. In addition, physical limitations of storage resources will imply the definition of a data management policy. Simple algorithms as LRU will be implemented in order to remove the most older data. This will avoid to overload the system.

Data Location When a data item has been sent once from a client to the platform, the data management service has to be able to find where data is stored to use it in other computations on other servers. Furthermore, in order to obtain a scalable infrastructure, we need to separate the logical view of the data from its physical location. Even if the solution of metadata [8] is elegant, a data management service in NES environments has not exactly the same characteristics than other data management systems implemented in Grid Computing Environments. In fact, in these environments, clients need to access huge data for analysis. Hence, these systems are built in order to provide a reliable access to data that are geographically distributed. In ASP environments, numerical applications to which NES platforms give access have generally data that are produced and directly accessed by the client that sends the request. ASP environments have to give a reliable access to computational servers even if the problematic of data access by clients is also a constraint. This is why it is not necessary to define data along their characteristics. Nevertheless, it is mandatory to fully identify data that are stored inside the platform.

Data Movement As seen above, a data management service in ASP environments is able to store and locate data. But when data is required for more than one computation on more than one server, it is also mandatory to be able to move data between computational servers. In fact, if we consider that time to transfer data between servers is smaller than time to transfer data between clients and servers, we need to define a data movement mechanism. Obviously, when data is moved from one server to an other computational server, information on its location have to be updated.

Persistence Mode A data can be stored inside the platform and moved between storage resources. But, have all data sent by clients or produced by servers to be stored inside the platform? For obvious performance motivations, it is better to limit data persistence to those that are really useful. We think that only clients know which data have to stay inside the system. Hence, this is why we define a persistence mode in the help of which clients can tell if their data should be stored or not.

Security Once data are stored inside the platform, we need to define a policy to make secured operations on data. In fact, data stored inside the platform can be shared between clients. However, all the clients of the platform are not able to realize all operations on all data. As data stored are identified inside the platform, only the client that has produced the data has to be informed of the identifier that has been bound to its data in order to use it for later computation requests. Moreover, in collaborative projects for example, a client may want to share its stored data with other researchers but he does not want them to delete its data. We propose to add an access key in addition to the identifier. Thus, if a client wants to get read/write rights on a specified data, he has to join this key to the data identifier. Indeed, if the client that has produced the data does not want the others to have write access on it, he just have to provide the identifier. This leaves the responsibility of the management of its own data to the client. Simple mechanisms such as *md5, sha1* algorithms or routines like *urandom* will be chosen to generate such a key.

Fault Tolerance The fault tolerance policy is directly linked to the consistency policy. In fact, our approach does not define fault recovery mechanisms but only a consistency mechanism of the infrastructure when faults occur. Thus, only a context/contents model is defined. We ensure that all operations (add, remove) made on data by clients are made such that all the infrastructure is consistent. If a component that manages the physical data fails (named DataManager), updates on the architecture are made. We distinguish two possible cases of fault. A component that manages the logical view of data fails (named LocManager) or DataManager fails. If a LocManager fails, all its subtrees are considered as lost. We only ensure that the parent of the LocManager removes all references of data referenced on this branch. If a DataManager fails, we ensure that all references of data owns by it are removed in the hierarchy. No data recovery is made. We also consider that all data transfers are realized in a correct way but we make sure that updates are realized only when transfers are complete. A solution will be to replicate data.

Data Sources Heterogeneity Generally, a data is sent from the local machine of a client. However, it is also possible that a client does not owns the data it wants to send to the platform but only knows its location. Hence, we propose to give the possibility for a client to inform the server to pull data from a remote storage depot that is extern to the platform. This model has to deal with the support heterogeneity. We have first developed a model that allow the use of ftp and http protocols. These models have to be completed to interact with other protocols such as gridFTP. This approach is quite similar to the Stork approach for multi-protocols data transfers presented in [14].

Replication One mandatory aspect of a data management service is to provide a data replication policy. In fact, the need of data replication is particularly required for parallel tasks that share

data. Thus, a data management service needs to provide an API in order to move or replicate data between computational servers. This API will be used by a task scheduler for example.

4.2 The DIET Data Tree Manager

The data management service we implemented is based on the principles defined above. In this section, we present our implementation.

4.2.1 The Persistence Mode

A client can choose whether a data will be persistent inside the platform or not. We call this property the `persistence mode` of a data. We have defined several modes of data persistence as shown in Table 1.

mode	Description
DIET_VOLATILE	not stored
DIET_PERSISTENT_RETURN	stored on server, movable and copy back to client
DIET_PERSISTENT	stored on server and movable
DIET_STICKY	stored and non movable
DIET_STICKY_RETURN	stored, non movable and copy back to client

Table 1: Persistence Modes.

4.2.2 The Data Identifier

When a data is stored inside the platform, an identifier is assigned to it. This identifier (also known as data handler) allows us to point out a data in an unique way within the architecture. It is clear that a client has to know this identifier in order to use the corresponding data. Currently, a client knows only the identifiers of the persistent data it has generated. It is responsible for propagating this information to other clients. Note that identifying data in NES environments is a relatively new issue. This is strongly linked to the way we are considering data persistence. In NetSolve, the idea is that data is persistent for a session time and deleted after. In DIET, we think that a data can survive to a session and could be used by other clients than the producer or in later sessions. Nevertheless, a client can also decide that its data are only available in a single session. Currently, as explained before, data identifiers are stored in a file in a client directory.

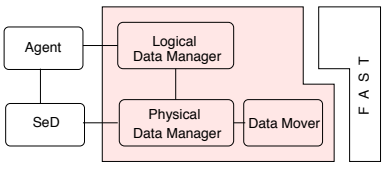


Figure 10: DTM: Data Tree Manager.

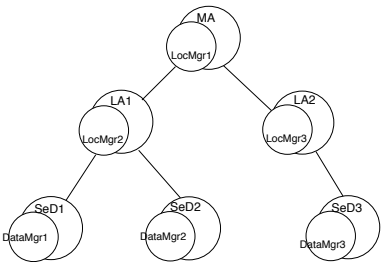


Figure 11: `DataManager` and `LocManager` objects.

4.2.3 Logical Data Manager and Physical Data Manager

In order to avoid interleaving between data messages and computation messages, the proposed architecture separates data management from computation management. The Data Tree Manager is build around two main entities.

Logical Data Manager The Logical Data Manager is composed of a set of `LocManager` objects. A `LocManager` is set onto the agent with which it communicates locally. It manages a list of couples (data identifier, owner) which represents data that are present in its branch. Hence, the hierarchy of `LocManager` objects provides the global knowledge of the localization of each data.

Physical Data Manager The Physical Data Manager is composed of a set of `DataManager` objects. The `DataManager` is located onto each SeD with which it communicates locally. It owns a list of `persistent` data. It stores data and has in charge to provide data to the server when needed. It provides features for data movement and it informs its `LocManager` parent of updating operations performed on its data (add, move, delete). Moreover, if a data is duplicated from a server to another one, the copy is set as non persistent and destroyed after it uses with no hierarchy update.

This structure is built in a hierarchical way as shown in Figure 11. It is mapped on the DIET architecture. There are several advantages to define such a hierarchy. First, communications between agents (MA or LA) and data location objects (LocManager) are local like those between computational servers (SeD) and data storage objects (DataManager). This ensures that this is not costly, in terms of time and network bandwidth, for agents to get information on data location and for servers to retrieve data. Secondly, considering the physical repartition of the architecture nodes (a LA front-end of a local area network for example), when data transfers between servers localized in the same subtree occur, the consequently updates of the infrastructure are limited to this subtree. Hence, the rest of the platform is not involved in the updates.

4.2.4 Data Mover

The Data Mover provides mechanisms for data transfers between Data Managers objects as well as between computational servers. The Data Mover has also to initiate updates of DataManager and LocManager when a data transfer has finished.

4.2.5 Client API

A client can specify the persistence mode of its data. This is done when the problem profile is build. Moreover, after the problem has been evaluated by the platform and persistent data are sent or produced, a unique identifier is affected to each data. A client can execute several operations using the identifier:

Data Handle Storage The `store_id()` method allows the data identifier to be stored in a local client file. This will be helpful to use data in other session for the same client or for other clients.

```
store_id(char *handle, char *msg);
```

Utilization of the data handle The `diet_use_data()` method allows the use of a data stored in the platform identified by its handle. The description of the data (its characteristics) is also stored.

```
diet_use_data(char *handle);
```

Data Remove The `diet_free_persistent_data()` method allows to free the persistent data identified by `handle` from the platform.

```
diet_free_persistent_data(char *handle);
```

Read an already stored data The `diet_read_data(char * handle)` method allows to read a data identified by `handle` already stored inside the platform.

```
diet_data_t diet_read_data(char *handle);
```

5 Experimental Results

5.1 Standard NetSolve Data Management

In this section we test the standard version of NetSolve. We first make experiments on NetSolve without data management and then with the two NetSolve data management approaches described in Section 2.1.2: the Distributed Storage Infrastructure, used to provide data transfer and storage and the Request Sequencing used to decrease network traffic amongst client and servers.

Experiments Servers are distributed on a site far from approximatively 100 kilometers to the client. Wide area network is a 16 Mbits/s network while the local area network is an Ethernet 100 Mbits/s network. The platform built for NetSolve tests is composed of three servers, an agent, and an IBP depot.

The experiments consist in a sequence of calls in a session: $C = A * B$ then $D = C + E$ then $A =^t A$. We made three series of test for NetSolve. First, a test using three consecutive blocking calls. Then, a request sequencing test and finally a test with DSI. The last test is divided into two parts: first, a single server computes all the sequence, then each call is computed by a different server.

Results of the series of tests are exposed in Figure 12. We note that Request Sequencing is the best solution for such a sequence of calls. When using DSI, we note also that the best solution is when three servers are involved in the computation. This is a bit surprising but it is confirmed by different others tests we made building several different topologies (a server that is also an IBP depot, an IBP depot closest from one server than for the others). In fact, in order to confirm this fact, we try to choose the best server (in terms of processing power and memory capacity) that compute the three calls: but the best solution is always when three servers were involved. We can explain this fact by the memory limitations of the servers involved. A server that have to process three computations does not free its memory implying an overload of this server for further computation.

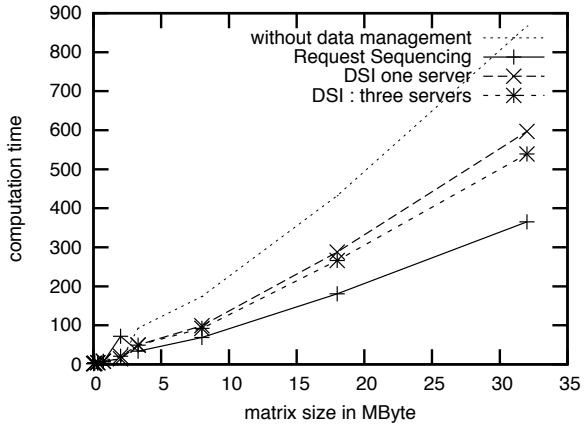
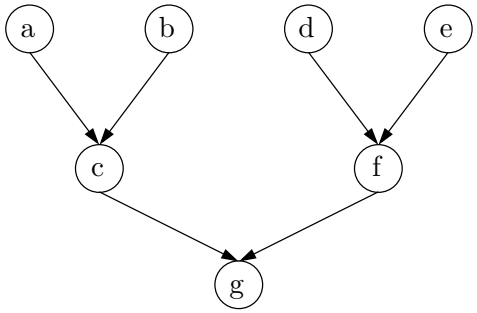


Figure 12: Standard NetSolve tests.




```
c=a*b
f=d*e
g=c*f
```

Figure 13: Matrix multiplication program task graph.

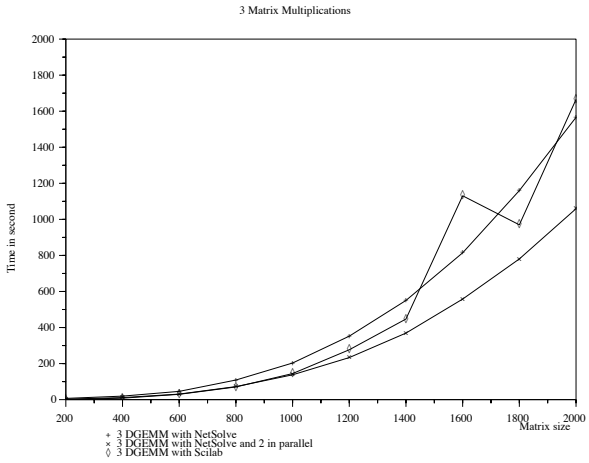


Figure 14: Matrix multiplications using NetSolve with data persistence on a LAN.

5.2 NetSolve with Data Persistence and Redistribution

In this section we show several experiments that demonstrate the advantage of using data persistence and redistribution within NetSolve as described in Section 3. Figures 14 and 16 show our experimental results using NetSolve as a NES environment for solving matrix multiplication problems in a grid environment.

5.2.1 LAN experiments

In Figure 14, we ran a NetSolve client that performs 3 matrix multiplications using 2 servers. The client, agent, and servers are in the same LAN and are connected through Ethernet. Computation and task graphs are shown in Figure 13. The first two matrix multiplications are independent and

can be done in parallel on two different servers. We use Scilab[5] as the baseline for computation time. We see that the time taken by Scilab is about the same than the time taken using NetSolve when sequentializing the three matrix multiplications. When doing the first two ones in parallel on two servers using the redistribution feature, we see that we gain exactly one third of the time, which is the best possible gain. These results show that NetSolve is very efficient in distributing matrices in a LAN and that non-blocking calls to servers are helpful for exploiting coarse grain parallelism.

5.2.2 WAN Experiments

$$C_{11} = A_{11}B_{11} \; ; \; C_{22} = A_{21}B_{12}$$

$$C_{12} = A_{11}B_{12} \; ; \; C_{21} = A_{21}B_{11}$$

$$C_{11} = C_{11} + A_{12}B_{21} \; ; \; C_{22} = C_{22} + A_{22}B_{22}$$

$$C_{12} = C_{12} + A_{12}B_{22} \; ; \; C_{21} = C_{21} + A_{22}B_{21}$$

Figure 15: Matrix multiplication using block decomposition.

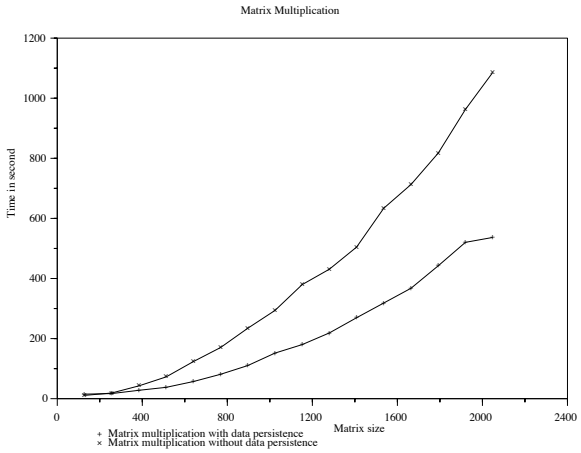


Figure 16: NetSolve with data persistence: WAN experiments

We have performed a blocked matrix multiplication (Figure 15). The client and agent were located in one University (Bordeaux) but servers were running on the nodes of a cluster located in Grenoble[6]. The computation decomposition done by the client is shown in Figure 16. Each matrix is decomposed in 4 blocks, each block of matrix A is multiplied by a block of matrix B and contributes to a block of matrix C. The first two matrix multiplications were performed in parallel. Then, input data were redistributed to perform matrix multiplications 3 and 4. The last 4 matrix multiplications and additions can be executed using one call to the level 3 BLAS routine DGEMM and requires input and output objects to be redistributed. Hence, this experiment uses all the features we have developed. We see that with data persistence (input data and output data are redistributed between the servers and do not go back to the client), the time taken to perform the computation is more than twice faster than the time taken to perform the computation without data persistence (in that case, the blocks of A, B, and C are sent back and forth to the client). This experiment demonstrates how useful the data persistence and redistribution features that we have implemented within NetSolve are.

[5] www.scilab.org

[6] Grenoble and Bordeaux are two French cities separated by about 800 km.

5.3 DIET Data Management

The first experiments consist in a sequence of calls in a session: $C = A * B$, $D = C + E$ and $A =^t A$. The DIET platform is composed of one MA, two LAs and three servers. Servers are distributed on a site far from approximatively 100 kilometers from the client. The wide area network is a 16 Mbits/s network while the local area network is an Ethernet 100 Mbits/s network. Computers (0.5 Ghz up to 1.8 Ghz) are heterogeneous and run the Linux operating system. We conducted three series of tests: first, a test using three synchronous calls without using DTM. Then, the same sequence using DTM (i.e. using persistence): in this way, A, B, and E matrices are defined as persistent, C matrix must be persistent because it is an input data for the second problem. D matrix can be non persistent because it is not used anywhere else after. Hence, for this case, A, B, E are sent once, and C is not sent. For the last test, only identifiers are sent since all data are already present in the infrastructure.

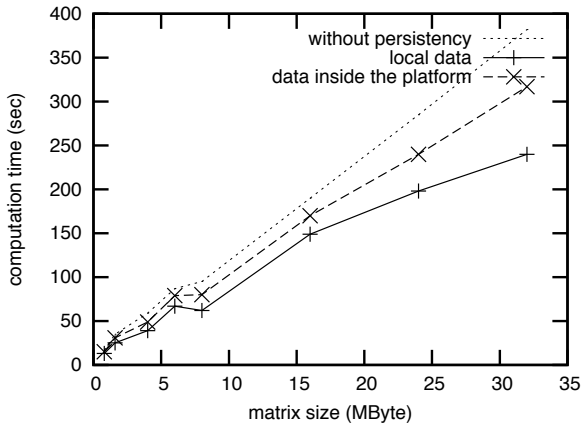


Figure 17: DIET Tests with and without persistence.

Results of the series of tests are exposed in Figure 17. If we can avoid multiple transmissions of the same data, the overall computational time is equal to the transfer time of data into the infrastructure plus the tasks computation time plus the results transfer time to the client. Unsurprisingly again, the last scenario appears to be the best one and confirms the feasibility and the low cost of our approach in the case of a sequence of calls. Using the CORBA space, we can avoid the copy of data by using CORBA memory management methods. These methods allow to get a value without making a memory copy. Moreover, notice that the update of the hierarchy is performed in an asynchronous way, so its cost is very small and does not influence the overall computational time. However, for large data, this approach has the limitations of the memory management.

To complete experiments already lead in [5] and the above results, we have conducted series of tests in order to show the overall advantages of using persistence in DIET. This target architecture is composed of one MA, two LA and two SeD located in a local network. A client is located in a remote site far from 100 kilometers to DIET. The wide area network is a 16 Mbits/s network while the local area network is an Ethernet 100 Mbits/s network. The deployed application is a linear algebra application in which computation time is relatively independent from data size.

In the first experiment, data are in input mode. As seen in Figure 18, the time of execution varies enormously according to the case. When data is persistent and locally stored onto the

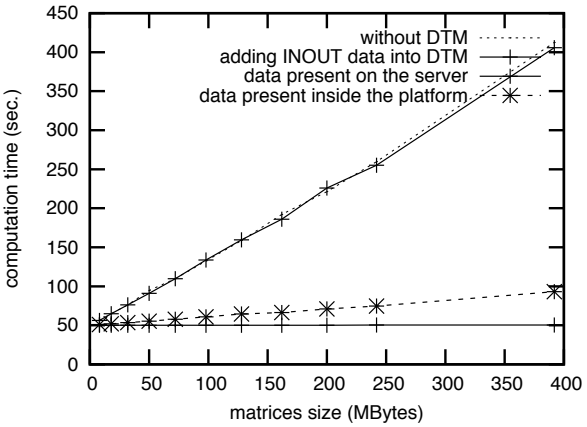


Figure 18: Sending IN data.

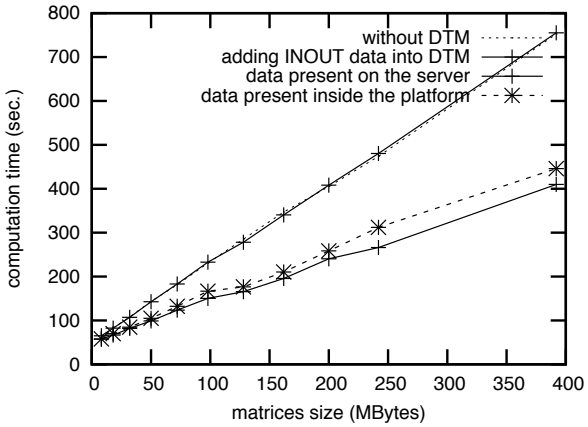


Figure 19: Sending INOUT data.

computational server, the global execution time is equal to the application computation time. This difference corresponds to the data transfer time profit: approximately 87% for a 400 MBytes matrix. When data is moved between computational servers the gain is of an order of 77% for a 400 MBytes matrix. The difference in gain corresponds to the data transfer time.

In the second experiment, the mode of data is inout. Profits are less important than for the first experiment, as shown Figure 19: approximatively 45% for a 400 MBytes matrix if the data is local to the computational server and 40% if the data is moved.

These results confirm the feasibility of our approach and the gains in term of execution time.

5.4 DIET and NetSolve Comparison

We summarize here the differences between standard NetSolve, NetSolve with data persistence and redistribution (called NetSolve-PR here), and DIET with data management.

- In standard NetSolve request sequencing approach, the sequence of computations has to be processed by an unique server. In this case, a client needs to have the knowledge of the services provided by a server in order to use this approach. Now, when using DSI, it is useful to have a DSI depot near computational servers in order to decrease transfer time. Hence, the way that DSI architecture is implemented is very important. In NetSolve-PR and in DIET DTM, a client does not need to know which server is able to solve a given problem (considering that a submitted request can be processed by the platform), and we assume that the data management architecture allows data to be close to the computational server.

- The DIET Data Mover is directly managed by the DTM that allows data to be moved near computational servers. In standard NetSolve with DSI, considering for example two far away computational servers that will need the same data, data must be sent on a DSI depot that is close to each computational server. Hence, data could be sent twice by a client. In NetSolve-PR data always stay on a server and do not use a depot. Data can be sent directly from a client to a server.

- Using NetSolve approach, a client does not need to specify the way its data will be managed. Using request sequencing or DSI, data are considered to be persistent. In DIET DTM,

users need to precise the persistence mode of all their data, even for the non persistent ones. NetSolve-PR is backward compatible. This means that when persistence is not needed nothing as to be specified. However, when using persistence the client has to specify it in the request.

- In DIET, we think that persistent data must "survive" to a client session and so must be fully identified. Data are kept as long as a client needs it (for later use in other sessions and for other clients in case of collaborative projects for example). In NetSolve (with or without data persistence and redistribution), data are persistent in a session, for a set of computations: data are lost when the client terminate.

- In NetSolve, the system cannot be overloaded by data since data are removed from its depot after computation or removed after a set of computation (request sequencing). In DIET and NetSolve-PR, the way data is managed may lead to a memory overload since data is cached on servers when they are not explicitly send as files.

6 Standardizing Data Management

As we seen, data management in ASP environments leads to several approaches. However, the need of a common API for ASP environments is essential. Indeed, NetSolve, Ninf and DIET are members of the GridRPC working Group in the GGF which work is to standardize and to implement a remote procedure call mechanism for Grid Computing. This work has already lead to a programming model [20].

Within this GridRPC working group an on-going work supervised by Craig Lee aims at standardizing data management for this model. So far, the proposal is based on two points: a data must be fully identified and a programmer can choose whether a data will be persistent inside the platform or not. This proposal must take into account the different approaches in ASP environments in order to obtain a common layer on which each policy can be integrated.

In order that each data will be fully identified, we define the **data handle** (DH) which is the reference of a data that may reside anywhere. This enables the virtualization of data since it can be read or written without knowing or caring where it is coming from or going to. The creation of a **data handle** is realized by the `create(data_handle_t *dh);` function.

Once the data reference created, it is also possible to bind it with a data. If data is bound, it must be on the client or on a storage server. Otherwise, data is already stored inside the platform. The bind operation is also used to specify if the data must be keep or not. This operation is realized by the `bind(data_handle_t dh, data_loc_t loc, data_site_t site);` function.

- **data_loc_t loc** (data location): client side or storage server.

- **data_site_t site**: location of the machine where data will be stored If (site == NULL) data will be stored on the last computational server (client transparent) If (site == loc) data forwarded to site (client or storage server) If (site <> loc) data moved from loc to site.

From these to functions, we can define operations on data handles.

- **data_t read(data_handle_t dh)**: read (copy) the data referenced by the DH from whatever machine is maintaining the data. Reading on an unbound DH is an error.

- **write(data_t data, data_handle_t dh)**: write data to the machine, referenced by the DH, that is maintaining storage for it. Writing on an unbound DH could have the default semantics of binding to the local host. This storage does not necessarily have to be pre-allocated nor does the length have to be known in advance.
- **data_arg_t inspect(data_handle_t dh)**: Allow the user to determine if the DH is bound, what machine is referenced, the length of the data, and possibly its structure. Could be returned as XML.
- **bool free_data(data_handle_t dh)**: free the data (storage) referenced by the DH.
- **bool free_handle(data_handle_t dh)**: frees the DH.

Figure 20 shows an example of data management within this proposed framework. In this figure, a client submits a problem to a server that is able to compute it and a second problem on an other server. The second server has best performance. For this second computation, the client have not to send data an other time, this data is already in the network.

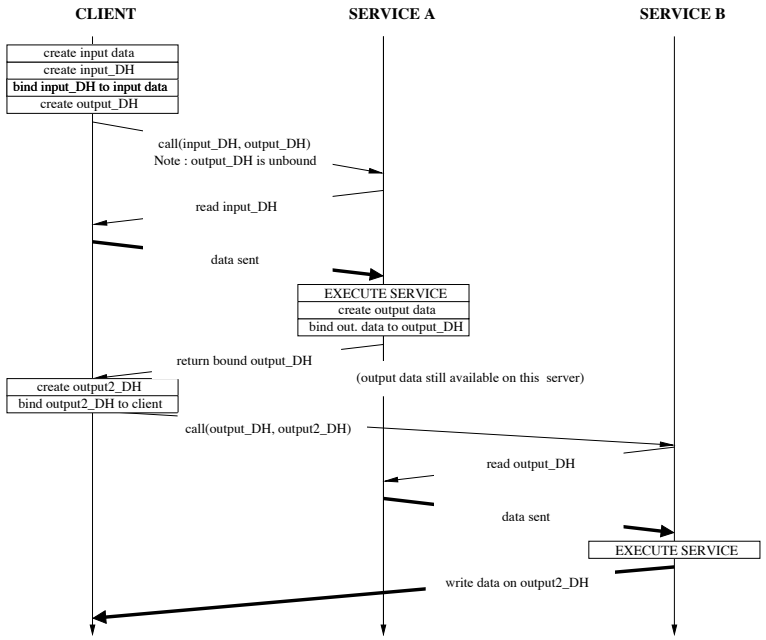


Figure 20: Using the GridRPC API for data management.

7 Conclusion and Future Work

The litterature proposes several approaches for executing applications on computational grids. The GridRPC standard implemented in several NES middleware (DIET, NetSolve, Ninf, etc.) is one of most popular paradigm. However, this standard does not define how data can be managed by the system: each time a request is performed on a server, input data are sent from the client to the server and output data are sent back to the client and thus data are not persistent. This implies

a large overhead that needs to be avoided. Moreover, no redistribution of persistent data between servers is available. When a data is computed by one server and needed by an other server for the next step of computation it always goes through the client, increasing the transfer time.

In this paper, we have proposed and implemented data management features in two NES (DIET and NetSolve). In NetSolve we changed the internal protocol in order to allow data to stay on server and to move data from one server to an other. We modified the API in order clients to allow data persistence and redistribution and we enhanced the request scheduling algorithm in order to take into account data location. Concerning DIET, we developed a data management service called Data Tree Manager (DTM). This service is based on three key points: a data must be fully identified inside the platform, it must be located and moved between computational servers. The way to think this service was relatively a new concept in NES community. Indeed, our service is able to keep information on data stored as long as the client does not want to remove them.

In our experimental results, we tested our implementations and the standard NetSolve one (which features request sequencing). We shown that data management improves the performance of applications (for both systems) when requests have dependences because it reduces the amount of data that circulates on the Network.

Since we show that the implementation of data management is feasible and it provides an increase of performance, we discuss, in the last section of this article, the standardization proposal (joint work with C. Lee within the GGF) of such a feature. It is based on two points: data is fully and globally identified, and the programmer can choose whether a data is persistent or not in an explicit way.

In our future work, we want to study and propose new scheduling algorithms that efficiently takes into account data management. For instance we believe that a better scheduling algorithm than the proposed enhancement of MCT can be designed in this context.

In the context of DIET, the overview of NetSolve DSI policy leads us thinking about the possibility to keep data on storage servers. The definition of an efficient storage policy will allow to avoid servers overload. Our idea is to keep data onto a server as long as it does not decrease server performance. The data will then be stored in available storage service systems (like IBP).

References

[1] P. Arbenz, W. Gander, and J. Moré. The remote computational system. *Parallel Computing*, 23(10):1421–1428, 1997.

[2] D.-C. Arnold, D. Bachmann, and J. Dongarra. Request Sequencing: Optimizing Communication for the Grid. In *Euro-Par 2000 Parallel Processing, 6th International Euro-Par Conference*, volume volume 1900 of Lecture Notes in Computer Science, pages 1213–1222, Munich Germany, August 2000. Springer Verlag.

[3] E. Caron, S. Chaumette, S. Contassot-Vivier, F. Desprez, E. Fleury, C. Gomez, M. Goursat, E. Jeannot, D. Lazure, F. Lombard, J.M. Nicod, L. Philippe, M. Quinson, P. Ramet, J. Roman, F. Rubi, S. Steer, F. Suter, and G. Utard. Scilab to Scilab$_{//}$, the OURAGAN Project. *Parallel Computing*, 27(11), 2001.

[4] E. Caron and F. Desprez. DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid. *International Journal of High Performance Computing Applications*, 2005. To appear. Also available as INRIA Research Report RR-5601.

[5] E. Caron, F. Desprez, B. Del-Fabbro, and A. Vernois. Gestion de données dans les nes. In *DistRibUtIon de Données à grande Echelle. DRUIDE 2004*, Domaine du Port-aux-Rocs, Le Croisic. France, may 2004. IRISA.

[6] Eddy Caron, Frédéric Desprez, Franck Petit, and Cédric Tedeschi. Resource Localization Using Peer-To-Peer Technology for Network Enabled Servers. Research report 2004-55, Laboratoire de l'Informatique du Parallélisme (LIP), December 2004.

[7] H. Casanova and J. Dongarra. NetSolve: A Network-Enabled Server for Solving Computational Science Problems. *International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212 – 213, Fall 1997.

[8] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets, 1999. http://www.globus.org/, 1999. 132.

[9] S. Dahan, J.M. Nicod, and L. Philippe. Scalability in a GRID server discovery mechanism. In *10th IEEE Int. Workshop on Future Trends of Distributed Computing Systems, FTDCS 2004*, pages 46–51, Suzhou, China, May 2004. IEEE Press.

[10] B. Del-Fabbro, D. Laiymani, J. Nicod, and L. Philippe. Data management in grid applications providers. In *Procs of the 1st IEEE Int. Conf. on Distributed Frameworks for Multimedia Applications, DFMA'2005*, pages 315–322, Besançon, France, February 2005.

[11] Bruno Del-Fabbro, David Laiymani, Jean-Marc Nicod, and Laurent Philippe. A data persistency approach for the diet metacomputing environment. In Hamid R. Arabnia, Olaf Droegehorn, and S. Chatterjee, editors, *International Conference on Internet Computing*, pages 701–707, Las Vegas, USA, June 2004. CSREA Press.

[12] GridRPC Working Group. `https://forge.gridforum.org/projects/gridrpc-wg/`.

[13] S. Sekiguchi H. Nakada, M. Sato. Design and Implementations of Ninf: Towards a Global Computing Infrastructure. *Future Generation Computing Systems, Metacomputing Issue*, 15:649–658, 1999.

[14] T. Kosar and M. Livny. Stork: Making data placement a first class citizen in the grid, 2004. In Proceedings of the 24th Int. Conference on Distributed Computing Systems, Tokyo, Japan, March 2004.

[15] M. Maheswaran, S. Ali, H.J. Siegel, D. Hengsen, and R.F. Freund. Dynamic Matching and Scheduling of a class of Independent Tasks onto Heterogeneous Computing System. In *Proceedings of the 8th Heterogeneous Computing Workshop (HCW '99)*, april 1999.

[16] S. Matsuoka, H. Nakada, M. Sato, , and S. Sekiguchi. Design Issues of Network Enabled Server Systems for the Grid. http://www.eece.unm.edu/~dbader/grid/WhitePapers/satoshi.pdf, 2000. Grid Forum, Advanced Programming Models Working Group whitepaper.

[17] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. A GridRPC Model and API for End-User Applications, December 2003. https://forge.gridforum.org/projects/gridrpc-wg/document/GridRPC_EndUse%r_16dec03/en/1.

[18] NEOS. `http://www-neos.mcs.anl.gov/`.

[19] Martin Quinson. Dynamic performance forecasting for network-enabled servers in a meta-computing environment. In *International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS'02), in conjunction with IPDPS'02*, April 15-19 2002.

[20] K. Seymour, C. Lee, F. Desprez, H. Nakada, and Y. Tanaka. The End-User and Middleware APIs for GridRPC. In *Workshop on Grid Application Programming Interfaces, In conjunction with GGF12*, Brussels, Belgium, September 2004.

Wrekavoc: a Tool for Emulating Heterogeneity

Louis-Claude Canon[1]

[1]ESEO
4 rue Merlet de la Boulaye
BP 30926
49009 Angers cedex 01, France
louis-claude.canon@eseo.fr

Emmanuel Jeannot[2]

[2]Loria INRIA-Lorraine
Campus scientifique
54506 Vandœuvre les Nancy
France
emmanuel.jeannot@loria.fr

Abstract

Computer science and especially heterogeneous distributed computing is an experimental science. Simulation, emulation, or in-situ implementation are complementary methodologies to conduct experiments in this context. In this paper we address the problem of defining and controlling the heterogeneity of a platform. We evaluate the proposed solution, called Wrekavoc, with micro-benchmark and by implementing algorithms of the literature.

1. Introduction

Research on algorithms for heterogeneous platforms is a very active domain. It encompasses the fields of scheduling [12], load balancing [1], linear algebra [8], data redistribution [3], etc. Unfortunately heterogeneity makes problems harder to solve. In few cases polynomial algorithms are found, sometimes only approximation algorithms are proposed while in many cases no theoretical results are available. In the later case an experimental approach can be used to test or compare heuristics. In large-scale distributed heterogeneous systems, numerous parameters and complex interactions between resources make models mostly intractable. Moreover, even if in some cases theoretical results are found, an experimental approach on a real platform can also help in validating both the modeling and the algorithm. Since experimental evaluation is mandatory in algorithmic (for heterogeneous platform) research, several complementary methodologies have been proposed.

Simulators focus on a certain part of the platform and abstract the remaining of the system. Simulations enable reproducible experiments and allow to test a large set of platforms and experimental conditions. Examples of simulators designed to test and compare scheduling algorithms in large-scale distributed systems are Bricks [16], GridSim [4] or Simgrid [10]. It is out of the scope of this paper to compare the relative merits of these simulators and the reader is referred to [15] for further details and other simulators. Surprisingly very few studies on the comparison between simulation and real experiments have been conducted. The validation of Bricks was performed by incorporating NWS [18] into Bricks. Then, they run an applications within Bricks and a real environment and compare the behavior of NWS under both environments. Simgrid validation was done by comparing simulation and analytical results on a tractable scheduling problem.

In some situations complex behaviors and interactions between the distributed resources cannot be simulated. This is due to the difficulty to capture and extract all the factors that play a role during the execution of a given application (such as OS specific features: for instance process scheduling, or hardware special capabilities: for instance hyperthreading, cache memory, multi-core processors or runtime performance: for instance the different versions or flavors of MPI). In-situ experiments solve this problem by running a real software on a realistic platform. Typically, experiments on real-life heterogeneous platforms are made using the available workstations of a laboratory. However, these machine are often shared with other users making reproducibility of experiments hard to achieve. Recently experimental dedicated platforms have been proposed to tackle this problem (Das-2 [5], Grid-explorer [7], Grid'5000 [6], or Planet-Lab [14]). However, the degree of heterogeneity of these platforms is very low and fixed. This makes the evaluation of algorithms designed for heterogeneous environment very hard to con-

duct and results hard to extrapolate for other heterogeneous cases. In order to tackle this problem, Latsovetsky et al. have proposed in [9] a new approach that consists in comparing the efficiency of the heterogeneous solution of a problem with the homogeneous one. In this case, the homogeneous setting have the same aggregate performance of of the heterogeneous one. But, still the heterogeneity is not controllable.

Between simulation and real-life experiments stands emulation (*e.g.*, Microgrid [19]) which goal is to test real applications (as in real-life experiments) with less abstraction than with simulators. However, in Microgrid each program have to be linked against the Microgrid library that interprets system call leading to an increase of the execution time.

In this paper we address the problem of controlling the heterogeneity of a cluster. Our objective is to have a configurable environment that allows for reproducible experiments on large set of configurations using real applications with no emulation of the code. Given an homogeneous cluster, our proposed solution (called Wrekavoc) degrades the performance of nodes and network links independently in order to build a new heterogeneous cluster. Then, any application can be run on this new cluster without modifications. This paper describes this tool, and its evaluation with micro-benchmark and by implementing algorithms for heterogeneous environments.

The remaining of this paper is organized as follow. In section 2 wrekavoc goals, model and implementation are presented. In section 3 describes how to define and control the heterogeneity of a cluster. Experimental results and validation of wrekavoc are given in Section 4. We compare our approach with other solution in section 5. Finally we conclude the paper section 6

2. Wrekavoc

2.1. Design Goals

Given a homogeneous cluster[1] we want to transform it into an heterogeneous one. We also want heterogeneity to be controlled and reproducible. One way of transforming an homogeneous cluster into an heterogeneous one is to update the hardware with more powerful devices (upgrading the CPU, adding some memory, etc.) . However, in this case, the heterogeneity is fixed and not controllable. An other way is to degrades its performances. This is the approach taken in this paper because it leads to a controllable layout. We target the degradation of the following characteristics:

- CPU power,
- network bandwidth,
- network latency and
- memory.

The degradation of each characteristic has to be independent (one can degrades CPU power without modifying network performance) and by software means (without modifying each node or rebooting the cluster). Lastly, we want to be able to configure a large cluster easily and rapidly.

2.2. Implementation

Our solution (called Wrekavoc) is implemented using the client-server model. A server, with administrator privilege, is deployed on each node one wants to configure and run as daemon. The client reads a configuration file and sends orders to each node in the configuration. The client can also order the nodes to recover the original state. The overall software stack is described in Fig. 1.

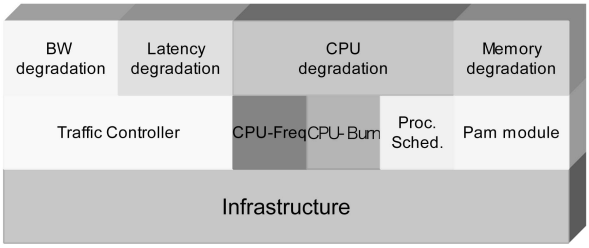


Figure 1. Software stack of Wrekavoc.

CPU Degradation. We have implemented several methods for degrading CPU performance. The first approach consists in managing the frequency of the CPU through the kernel CPU-Freq interface. This interface was designed to limit CPU frequency in order to save the power on laptops. It is based on several CPU technologies (such as *powernow*) which are not always available on cluster nodes. However, only 10 different frequencies are available through CPU-Freq. Therefore, if the required CPU technologies are not available on the nodes or if the discretization is too coarse we propose two other solutions. One is based on CPU burning. A program that runs under real-time scheduling policy burns a constant portion of the

[1] As a first approach we target Unix (preferably Linux) clusters.

CPU, whatever the number of processes currently running. More precisely, a CPU-burn sets the scheduler to a FIFO policy and gives itself the maximum priority. It then compute the time it needs to make a small computation. This computation is blocking and therefore no other program can use the CPU. After the computation the CPU burner then sleeps for the corresponding amount of time. It then restarts the whole process. A small tuning time is needed to make sure sleeping and calculation time are longer than 5ms, in order to be executed by the scheduler. The system call used to set the scheduler is `sched_setscheduler`. Thanks to the Unix `sched_setaffinity` system call, each CPU burner is tight to a given processor on a multi-processor node. The main drawbacks of this approach is that the CPU limitation occurs for every processes whatever its mode (kernel or user) and therefore, the network bandwidth is limited by the same fraction than the CPU. When an independent limitation of the CPU and the network is required, we propose a third alternative based on user-level process scheduling called CPU-lim. A CPU limiter is a program that supervises processes of a given user. Using the `/proc` pseudo-filesystem, it suspends the processes when they have used more than the required fraction of the CPU using the `SIGSTOP` and `SIGCONT`. This alternative is used for the experiments of Section 4.

Network Limitation. Limiting latency and bandwidth is done using *tc* (traffic controller) [17] based on *Iproute2* a program that allows advanced IP routing. With this tools it is possible to control both incoming and outgoing traffic. Furthermore, the latest versions (above 2.6.8.1) allows to control the latency of the network interface. An important aspect of tc is that it can alter the traffic using numerous and complicated rules based on IP adresses, ports, etc. We use tc to define a network policy between each pair of nodes. It raises scalability issue as each nodes as to implement $n-1$ rules with a configuration with n nodes. This issue will be discussed in the experimental section. Degradation of network latency and bandwidth is implemented using Class Based Queueing (CBQ): incoming or outgoing packets are stored into a queue according to the given quality of service before being transmitted to the TCP/IP stack. In order to work a kernel version above 2.6.8 is required and needs to be compiled with the `CONFIG_NET_SCH_NETEM=m` option.

Memory Limitation. Wrekavoc is able to limit the largest malloc a user can make. This is possible through the security module PAM. However, we have not been able to limit the whole memory usable by all the processes yet.

3. Configuring and Controlling Nodes and Links

The configuration of a homogeneous cluster is made through the notion of islet. An islet is a set of nodes that share similar limitation. Two islets are linked together by a virtual network which can also be limited (see figure 2).

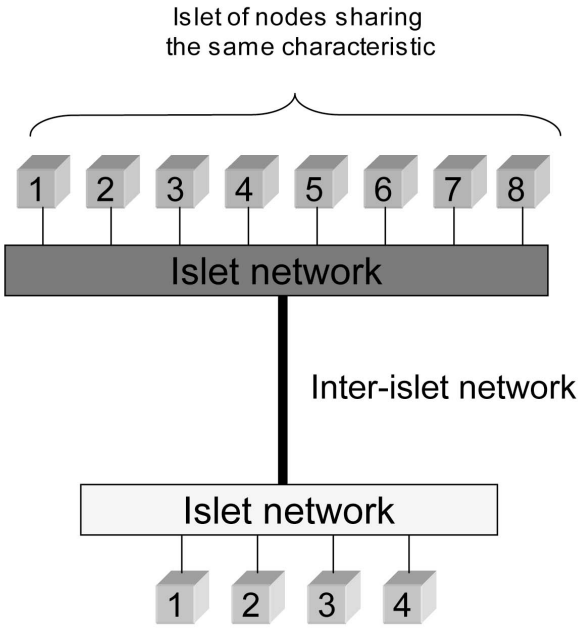


Figure 2. Islets logical view.

Defining an Islet. An islet is defined as a union of IP addresses (or machine names) intervals. The limitation parameters of each node of the islet take a value that can be defined in two ways. It can follow a Gaussian distribution[2] of the form [*mean*;*std. dev.*] or can follow a uniform distribution of the form [*min-max*]. For each islet we define several parameters. `SEED` is an integer that is used for the random distributions (-1 means a random seed). `CPU` is a distributed value of the CPU frequency in MHz of the noes of the islet. `BPOUT` (resp. `BPIN`) is a distributed value of the outgoing (resp. incoming) bandwidth in kB/s. `LAT` is the distributed value of network latency in ms. `USER` is the

[2]each node can have exactly the same value if we set 0 for the standard deviation.

```
islet1 :  [192.168.1.1-192.168.1.10] {
   SEED: 1
   CPU : [1000;0]
   BPOUT : [125000;0]
   BPIN : [125000;0]
   LAT : [0.05;0]
   USER : user1
   MEM : [80000;0]
}
islet2 :  [192.168.2.1-192.168.2.10]-[192.168.3.1-192.168.3.10] {
   SEED : -1
   CPU :[100-2000]
   BPOUT : [12500;0]
   BPIN : [12500;0]
   LAT : [0.05;0]
   USER : user1
   MEM :[80000;0]
}
!INTER : [islet1;islet2] [1250;0] [2500;0] [120;0] 1
```

Figure 3. Configuring two islets

name of the user for which the limitation are made. `MEM` is the distributed value of the maximum `malloc` in kB.

Linking Islets Together. Each islet configuration is stored into a configuration file. At the end of this file is described the network connection (bandwidth and latency) between each islet using the `!INTER` keyword. The last number of the `!INTER` line is the seed used for the random distribution.

Example. Figure 3 shows how to emulate two clusters within two islets. Of course, this configuration have to be executed on a cluster having at most the required performance. In this example, *Islet1*, is made of 10 nodes at 1 GHz with giga-ethernet interconnect (1Gb/s (125000 kB/s) bandwidth and 50 μs latency) and *Islet2* comprises 20 heterogeneous nodes with frequency between 100 MHz and 2GHz and a fast ethernet network (bandwidth: 100 Mbit/s, latency: 50 μs). The two clusters are linked by a network with a bandwidth of 10 Mbit/s and a latency of 120 ms from islet1 to islet2 and a bandwidth of 20 Mbit/s from islet2 to islet1. Remark that using -1 as seed for islet2 means that each time we configure the nodes we will get a different configuration of the nodes (reproducibility of the nodes configuration is done by using positive seeds).

4. Validation and Experimentation

All the experiments performed in this section were done using the Grid-explorer [7] cluster with 216 nodes. Each node has two 2 GHz AMD Opteron 246 with 2 GB of RAM. It runs under Linux Debian 3.1 with kernel 2.6.8.

4.1. Deployment Time.

The client reads the configuration file, parses the file and builds an XML file for each nodes. Then, it sends this sub-configuration to each node. When a node receives a configuration file it configures its own characteristics according to this file (see figure 4)

Figure 5 shows the configuration time against the number of nodes. 4 curves are shown: one islet, two islets, two nodes per islet and one node per islet. Results show that the configuration time increases with the number of nodes. The worst case occurs when we have one node per islet (the same number of islets as nodes). Even in this case configuring 130 nodes takes only 22 seconds, while with two nodes per islets it takes less than 10 seconds.

4.2. Micro-benchmark.

We have tested separately each kind of degradation using micro-benchmark.

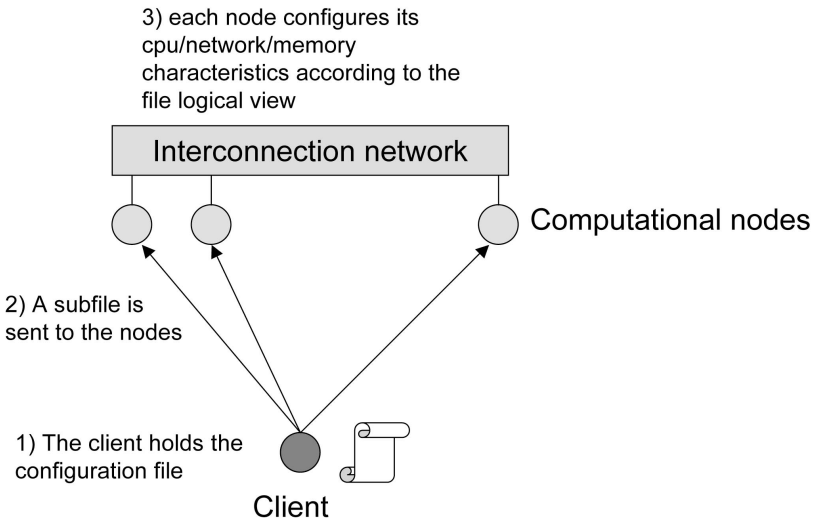


Figure 4. Steps for configuring a cluster

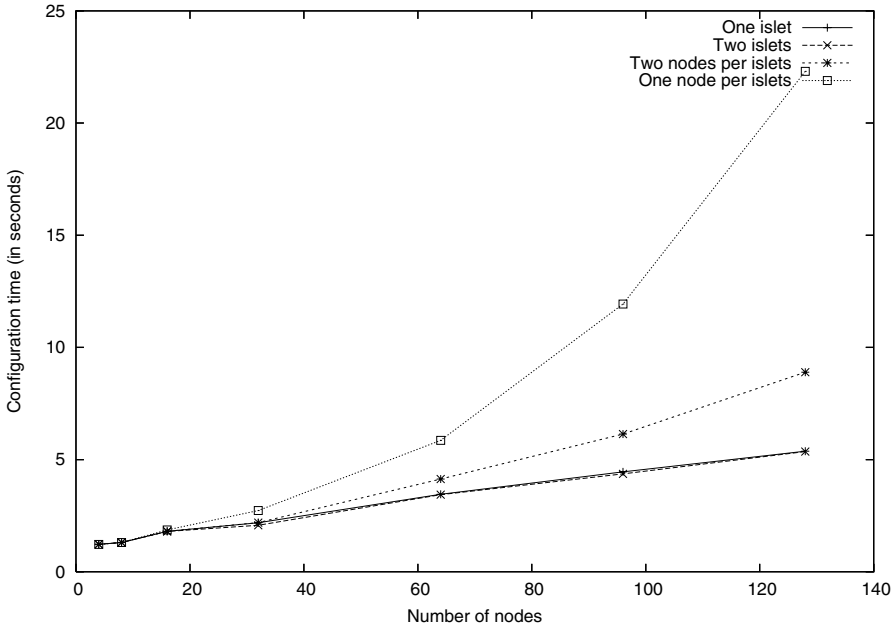


Figure 5. Configuration time for different islet sizes.

set latency	1	5	10	50	100	500	1000
RTT	2.12	10.05	20.12	100.058	200.20	1000.05	1999.75

Table 1. Round-trip-time against desired latency in ms.

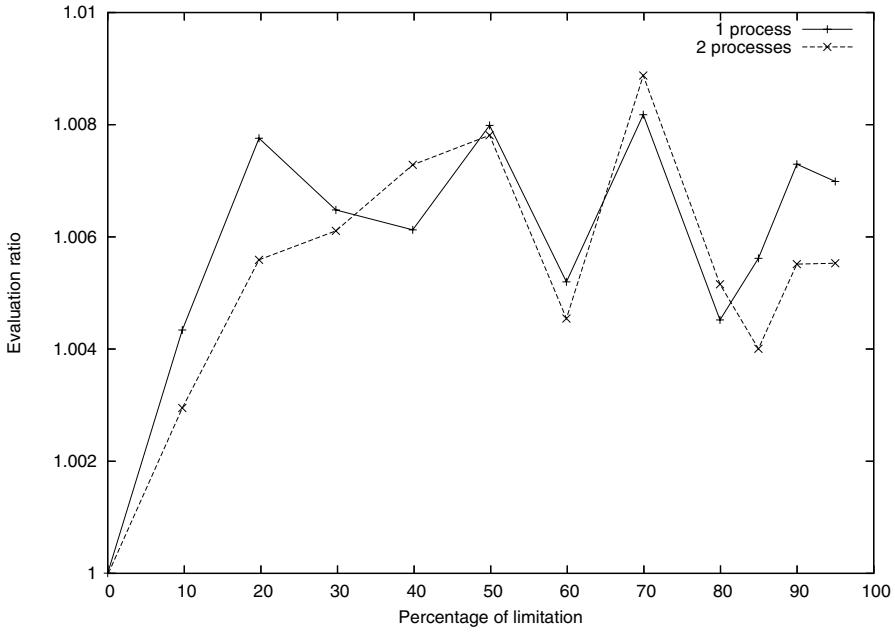


Figure 6. CPU micro-benchmark.

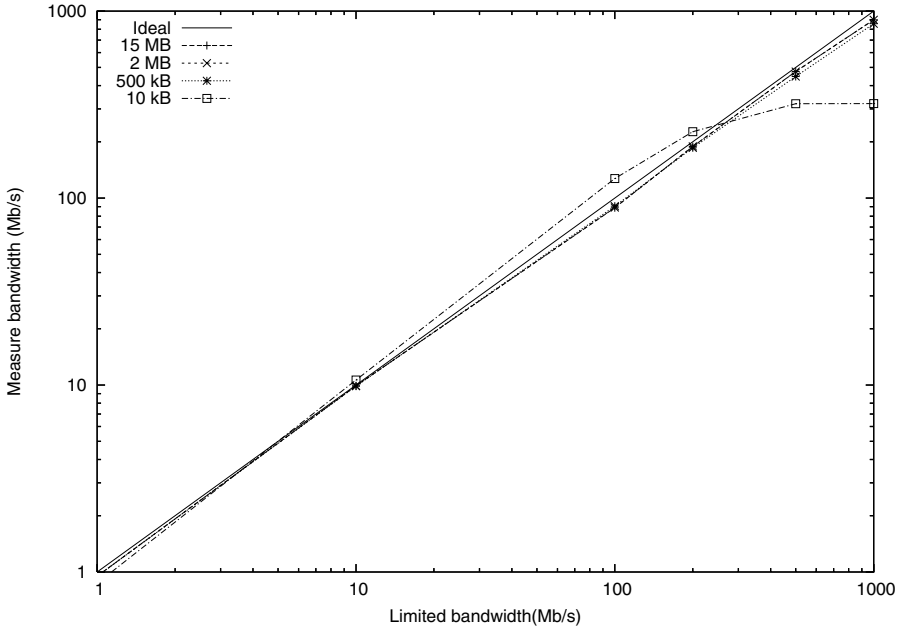


Figure 7. Bandwidth micro-benchmark.

CPU Tests. To measure how performance degradation impacts on the execution of a computation, we use the ratio between the expected and the actual duration times. The expected duration time is computed by applying the percentage of degradation to the time without degradation. When this ratio equals 1 this means that execution times matches the expected time. We can see on Figure 6 that the absolute value of this ra-

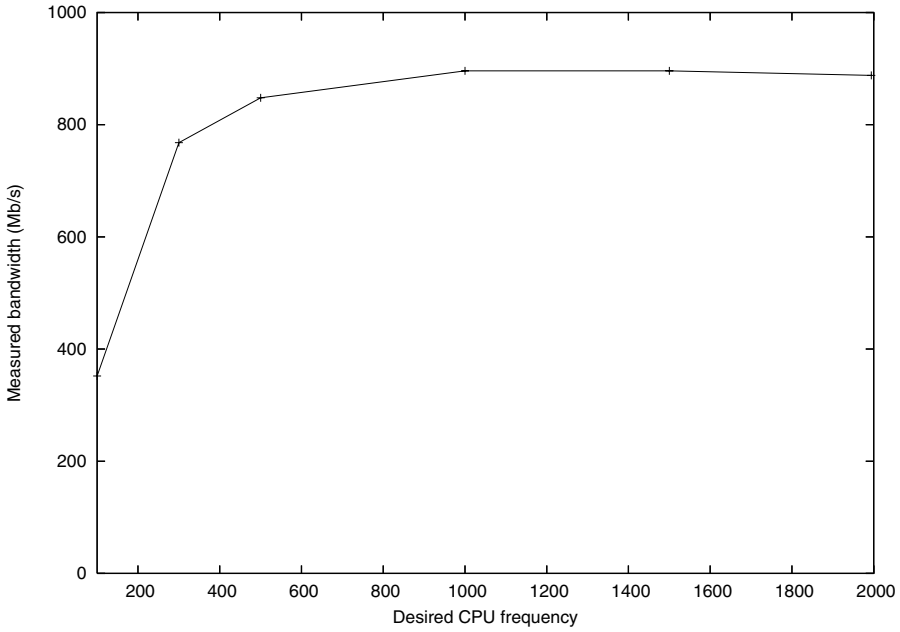


Figure 8. Testing the impact of CPU degradation against available bandwidth between two nodes linked by Gb Ethernet.

tio is never above 1.01 (i.e. less than 1% of difference). Therefore the CPU limitation behaves as expected and is able to handle bi-processors node.

Bandwidth Tests. Figure 7 shows the obtained bandwidth versus the desired bandwidth for different data size (between 15 MB and 10 kB). The ideal line shows what one should obtain theoretically. The results show that the obtained bandwidth is always very close to the desired one. We see that for 10 kB the we obtain a slightly greater bandwidth than the limited bandwidth. This is due to the fact that TC use some bucket to limit the bandwidth. The limitation starts when the bucket is completely filled. The amount of packets to fill the bucket being fixed, we see, for small messages that the real bandwidth is a little bit higher than the desired one. For this same size of data, we see that it is not possible to achieve the peak bandwidth. This is the same phenomena that happen in real network. Indeed, further investigations have shown that we obtain exactly the same bandwidth (320 Mbit/s) for 1 Gbit/s network card without network degradation for 10 kB messages.

Latency Tests. Table 1, shows the average round-trip-time (RTT) obtained by the *ping* command with various degraded latency. Results show that the RTT is exactly twice the value of the desired latency which is exactly what one should obtained theoretically as the latency is paid twice when doing a round trip.

4.3. Independent degradation

We have tested how the bandwidth degradation interfere with the CPU speed: we exchanged a file at different bandwidth speed while running a CPU intensive process. In this case, we have not seen any impact of the variation of the bandwidth with regards to the execution time.

Figure 8 show the obtained bandwidth against the CPU frequency on a Giga-ethernet network. Under 500 MHz the available bandwidth decreases with the CPU frequency. This is due to the fact that Gbit bandwidth is not achievable with slow CPU. In order to confirm that, we have exchanged data between two old PCs (PII at 400 MHz and PIII at 550 MHz) with a Gb PCI ethernet card under linux kernel 2.6.12 at runlevel 1. In this case the best achievable bandwidth was 340 Mb/s with UDP or TCP.

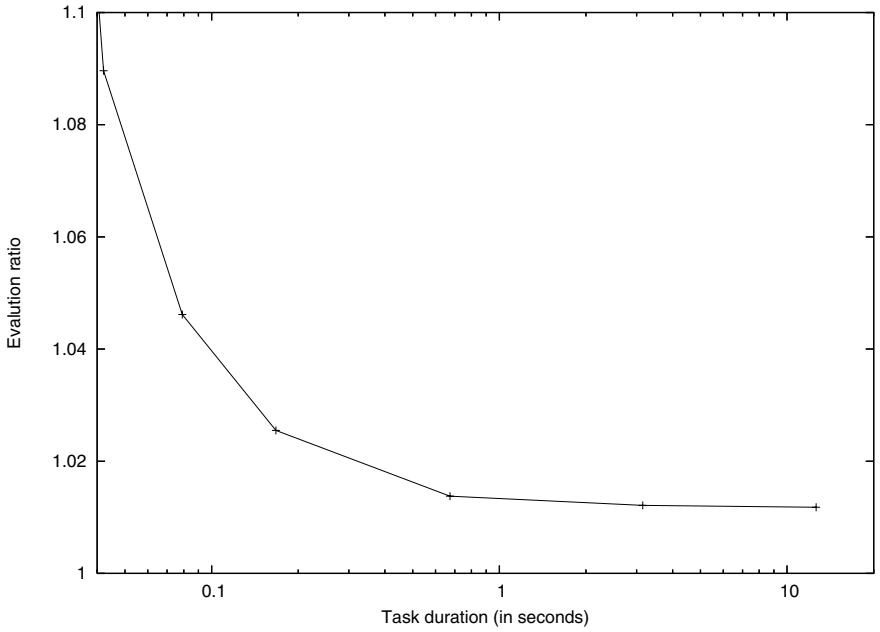


Figure 9. Testing a load-balancing algorithm.

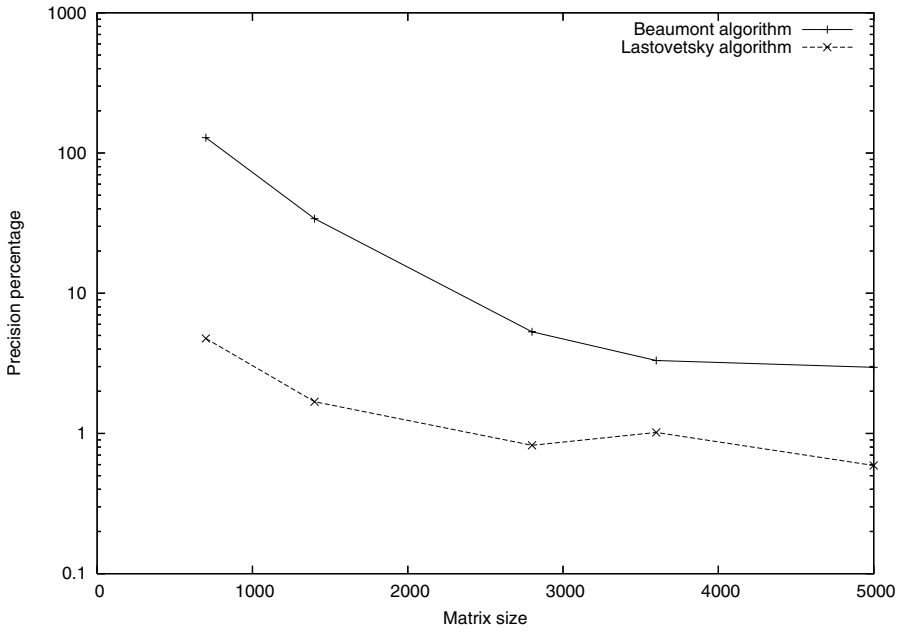


Figure 10. Testing two matrix multiplication algorithms.

4.4. Implementing Algorithms of the Literature.

We have tested Wrekavoc on several algorithms designed for heterogeneous environments.

First a static load-balancing algorithm of [11] page 160 and inspired from [13] is studied. We have chosen to balance a load composed of 500 tasks on 20 nodes when 5 nodes run at 100%, 5 at 75%, 5 at 50% and 5

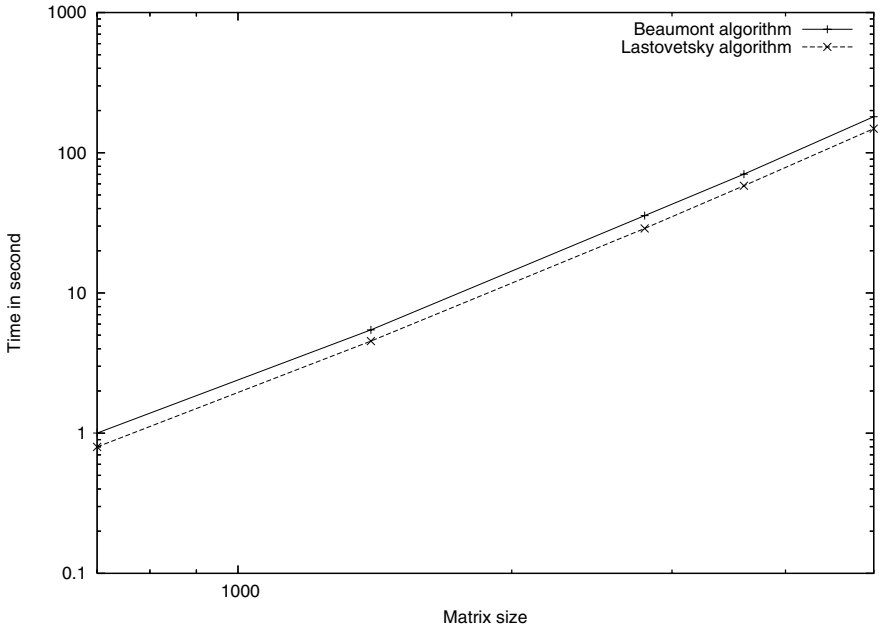


Figure 11. Comparing time of two matrix multiplication algorithms.

Methodology	Simulation (Simgrid)	Emulation (Microgrid)	In-Situ (hom. cluster)	L&R perf. analysis	Wrekavoc
Real app.	No	Yes	Yes	Yes	Yes
Abstraction	Very high	High	No	No	Low
Execution time	Speedup	Slowdown	Same	Same	Same
Proc. folding	Mandatory	Possible	No	No	No
Heterogeneity	Controllable	Controllable	No	Yes	Controllable

Table 2. Comparing different experimental methodologies

at 25%. For this simple case the algorithm balance the load inversely proportional to the speed of the machine (resp. 40 tasks, 30 tasks, 20 tasks and 10 tasks). Since the load balancing is perfect every node should theoretically finish at the same time. In Figure 9, on the y-axis, we plot the ratio $\frac{M}{m}$ where M is the finishing time of the last processor and m the finishing time of the first processor. This ratio is equal to 1 when all the processors finish at the same time. On the x-axis we plot the duration of one task on the 100% processor. Results show that the ratio is always under 1.1 and it decreases as the task duration increases. When task duration is greater than 0.1 second (on the fastest processor) the precision of the emulation is better than 5%.

Second we have run two matrix multiplication algorithms for heterogeneous environments. The first one of Beaumont et al. [2] is based on geometric partition of the column on the processors. The second one of Lastovetsky et al. [8] uses a data partitioning based on a performance model of the environment. Figure 10 shows the percentage $100 \times \frac{M-m}{m}$ (where M is the finishing time of the last processor and m the finishing time of the first processor) against the matrix size for both algorithms. We see that the emulation is correct as this percentage is under 5% for Lastovetsky algorithm. The fact that the percentage is worst for the Beaumont algorithm than for the Lastovetsky one means that the later algorithm provides a better load balancing with regards to the former one. We also provide the execution time of both algorithms for different matrix sizes in Figure 11. We see here that Lastovetsky algorithm outperforms the Beaumont one. However, the fact that these experiments shows that the

Lastovetsky algorithm outperforms the Beaumont algorithm should be consider as a side effect of this paper and not as a real contribution. Further studies such as implementation issues, degree of heterogeneity and so on, are required to assess or not this statement. What is shown here, is that Wrekavoc is a suitable tool for doing such comparison.

5. Related works

Studying and comparing heterogeneous parallel algorithms have been tackled by many researcher. Several tools and methodologies have been proposed to asses the performance of such algorithms. Several characteristics have to be defined in order to compare those approaches :

- the ability to execute a real application or just to simulate its execution.
- the level of abstraction of the target platform. The less abstraction, the better is the confidence in the obtained results.
- the speed of execution. Emulation tends to slowdown the execution while simulation tends to speed-up the execution time.
- the ability to fold several processors onto a single one. This enable the execution of a parallel application on only one processor.
- The management of heterogeneity. It is possible to have heterogeneity? Is it controllable?

In table 2 we compare simulation (i.e. simgrid [10]), emulation (i.e. microgrid [19]), in-situ (GdX [7]), Lastovseky and Reddy approach [9] and wrekavoc. We see that none of the proposed approaches have only advantages. This means that depending of the experimentation goal the methodology have to be chosen carefully.

6. Conclusion

Computer-science and especially heterogeneous distributed computing is an experimental science. Indeed, it is not always possible to obtain theoretical results for heuristics designed in this context. Moreover, it is very hard to derive tractable models of such environment.

Simulation (*e.g.*, Simgrid) emulation(*e.g.*, Microgrid) or in-situ implementation (*e.g.*, Grid 5000) are complementary methodologies that have been proposed to conduct experiments on heterogeneous platforms. They all present advantages and drawbacks. In this work we propose a new approach called Wrekavoc. The goal of Wrekavoc is to define and control the heterogeneity of a given platform by degrading CPU, network or memory capabilities of each node composing this platform. Our current implementation of Wrekavoc have been tested on an homogeneous cluster. We have shown that configuring a set of nodes is very fast. Micro-benchmarks show that we are able to independently degrade CPU, bandwidth and latency to the desired values. Tests on algorithms of the literature (load balancing and matrix multiplication) confirm the previous results and show that Wrekavoc is a suitable tool for developing, studying and comparing algorithms in heterogeneous environments.

Future works are directed toward applying this work to other platforms (such as Windows) or other kind of Unix. If CPU burning works for any cases, the other approaches and the network degradation needs to develop innovative solutions.

7. Acknowledgment

This work is partially funded by the *ACI Masses de données* "Data Grid Explorer" of the french Ministry of Research.

The authors would like to thanks the reviewers for very helpful comments and Marc Thierry for writing the first version of the code.

References

[1] J. M. Bahi, S. Contassot-Vivier, and R. Couturier. Coupling Dynamic Load Balancing with Asynchronism in Iterative Algorithms on the Computational Grid. In *IPDPS*, page 40, 2003.

[2] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert. Matrix Multiplication on Heterogeneous Platforms. *IEEE Trans. on Parallel and distributed Systems*, 12(10):1033–1051, Oct. 20001.

[3] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra. Block-cyclic redistribution over heterogeneous networks. *Cluster Computing*, 3(1):25–34, 2000.

[4] R. Buyya and M. M. Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *CoRR*, cs.DC/0203019, 2002.

[5] The DAS-2 project: `http://www.cs.vu.nl/das2/`.

[6] The Grid 5000 project: `http://www.grid5000.org/`.

[7] Grid explorer (GdX): `http://www.lri.fr/~fci/GdX/`.

[8] A. L. Lastovetsky and R. Reddy. Data Partitioning with a Realistic Performance Model of Networks of Heterogeneous Computers with Task Size Limits. In *ISPDC/HeteroPar*, pages 133–140, 2004.

[9] A. L. Lastovetsky and R. Reddy. On performance analysis of heterogeneous parallel algorithms. *Parallel Computing*, 30(11):1195–1216, 2004.

[10] A. Legrand, L. Marchal, and H. Casanova. Scheduling Distributed Applications: the SimGrid Simulation Framework. In *CCGRID*, pages 138–145, 2003.

[11] A. Legrand and Y. Robert. *Algorithmique Parallèle*. Dunod, 2004.

[12] M. Maheswaran, S. Ali, H. J. Siegel, D. A. Hensgen, and R. F. Freund. Dynamic Matching and Scheduling of a class of Independent Tasks onto Heterogeneous Computing System. In *Proceedings of the 8th Heterogeneous Computing Workshop (HCW '99)*, april 1999.

[13] F. Manne and T. Sørevik. Partitioning an Array onto a Mesh of Processors. In *PARA*, pages 467–477, 1996.

[14] Planet lab: `http://www.planet-lab.org/`.

[15] A. Sulistio, C. S. Yeo, and R. Buyya. A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. *Softw., Pract. Exper.*, 34(7):653–673, 2004.

[16] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms. In *HPDC*, 1999.

[17] iproute2+tc notes: `http://snafu.freedom.org/linux2.2/iproute-notes.html`.

[18] R. Wolski. Predicting CPU Availability on the Computational Grid Using the Network Weather Service. *Parallel Processing Letters*, 9(2):227–241, 1999.

[19] H. Xia, H. Dail, H. Casanova, and A. A. Chien. The MicroGrid: Using Online Simulation to Predict Application Performance in Diverse Grid Network Environments. In *CLADE*, page 52, 2004.

8. Biographies

Louis-Claude Canon was born in Angers, France. He received a secondary education diploma (major: mathematics and physics) in 2002. He is studying since 2002 in a French college (five-year curriculum) specialising in electronics and computer engineering (named ESEO). He is planning to complete this master and then to prepare a Ph.D. in computer science probably in a field related to the topic of the University of Angers (namely, artificial intelligence). During two trainings periods (in 2004 and in 2005), he got some experience in an academic environment. He realised his first contact with the research in Poona, in India, where he worked during two months for Dr. D. G. Kanhere on parallel programming and on a project which involved modeling and simulation. More recently, he assisted Pr. E. Jeannot in his research by pursuing the development Wrekavoc.

L.-C. Canon's research interests include parallel and distributed processing, modelling and simulation, language theory, algorithmic and artificial intelligence.

Emmanuel Jeannot is a full-time researcher at INRIA (Institut National de Recherche en Informatique et en Automatique) and is doing its research at the LORIA (Laboratoire Lorrain de Recherche en Informatique et ses Applications) in Nancy. He is currently invited at the Innovative Computing Laboratory of University of Tennesse, Knoxville. From sept. 1999 to sept. 2005 he was associate professor at the Université Henry Poincaré, Nancy 1. He got his PhD and Master degree of computer science (resp. in 1996 and 1999) both from Ecole Normale Supérieure de Lyon. His main research interests are scheduling for heterogeneous environments and grids, data redistribution, grid computing software, adaptive online compression and programming models. More informations are available at `http://www.loria.fr/~ejeannot`

GRID'5000: A LARGE SCALE AND HIGHLY RECONFIGURABLE EXPERIMENTAL GRID TESTBED

Raphaël Bolze[1]
Franck Cappello[2]
Eddy Caron[1]
Michel Daydé[3]
Frédéric Desprez[1]
Emmanuel Jeannot[4]
Yvon Jégou[5]
Stephane Lanteri[6]
Julien Leduc[2]
Noredine Melab[7]
Guillaume Mornet[5]
Raymond Namyst[8]
Pascale Primet[1]
Benjamin Quetier[2]
Olivier Richard[9]
El-Ghazali Talbi[7]
Iréa Touche[10]

Abstract

Large scale distributed systems such as Grids are difficult to study from theoretical models and simulators only. Most Grids deployed at large scale are production platforms that are inappropriate research tools because of their limited reconfiguration, control and monitoring capabilities. In this paper, we present Grid'5000, a 5000 CPU nation-wide infrastructure for research in Grid computing. Grid'5000 is designed to provide a scientific tool for computer scientists similar to the large-scale instruments used by physicists, astronomers, and biologists. We describe the motivations, design considerations, architecture, control, and monitoring infrastructure of this experimental platform. We present configuration examples and performance results for the reconfiguration subsystem.

Key words: Grid, P2P, experimental platform, highly reconfigurable system

The International Journal of High Performance Computing Applications,
Volume 20, No. 3, Fall 2006, pp. 481–494
DOI: 10.1177/1094342006070078

Figures 1–3, 6 appear in color online: http://hpc.sagepub.com

1 Introduction

Grid is well established as a research domain and proposes technologies that are mature enough to be used for real-life applications. Projects such as e-Science (http://www.nesc.ac.uk), TeraGrid (http://www.teragrid.org), Grid3 (http://www.ivdlg.org/grid2003), DEISA (http://www.deisa.org), and NAREGI (http://www.naregi.org/index_e.html/), demonstrate that large-scale infrastructures can be deployed to provide scientists with fairly easy access to geographically distributed resources belonging to different administration domains. Despite its establishment as a workable computing infrastructure, there are still many issues to be solved and mechanisms needed to optimize in performance, fault tolerance, QoS, security, and fairness.

As large-scale distributed systems, Grid software and architecture combine several characteristics which make them difficult to study by following a theoretical approach. Most of the research conducted in Grids is currently performed using simulators, emulators or production platforms. As discussed in the next section, all these tools have limitations making the study of new software and optimizations difficult. Given the complexity of Grids, there is a strong need for highly configurable real-life experimental platforms that can be controlled and monitored directly. Such tools already exist in other contexts. The closest example is PlanetLab (Chun et al. 2003). It consists of a set of PCs connected to the Internet and forming an experimental distributed system. PlanetLab is used for network studies as well as for distributed systems research.

In this paper we present the Grid'5000 http://www.grid5000.org project, still under construction but already in use in France. We first explain the motivation for developing a large scale, real-life experimental platform by discussing the limitations of existing tools. In Section 3, we present the design principles of Grid'5000 which were based on the results of Grid researchers' interviews.

[1]LIP, ENS LYON
[2]INRIA, LRI, PARIS (FCI@LRI.FR)
[3]INPT/IRIT, TOULOUSE
[4]LORIA, INRIA
[5]IRISA, INRIA
[6]INRIA SOPHIA ANTIPOLIS
[7]LIFL, UNIVERSITÉ DE LILLE
[8]LABRI, UNIVERSITÉ DE BORDEAUX
[9]LABORATOIRE ID-IMAG
[10]LGC, TOULOUSE

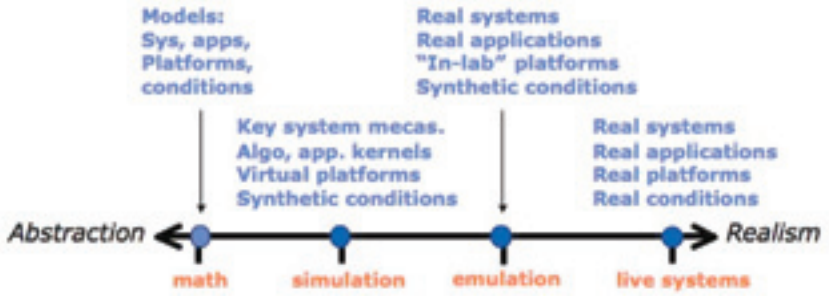


Fig. 1 Methodologies used in distributed system studies.

The implementation of Grid'5000 is described in Section 4. In Section 5 we present evaluation results for the deployment and reboot system, a key component of Grid'5000. Section 6 gives some configuration examples, demonstrating the high reconfigurability of the platform.

2 Motivations and Related Work

As with other scientific domains, research in Grid computing is based on a variety of methodologies and tools. Figure 1 presents the spectrum of methodologies used by researchers to study research issues in distributed systems. In large distributed systems, numerous parameters must be considered and complex interactions between resources make analytical modeling impractical. Thus simulators, emulators, and real platforms are preferred.

Simulators focus on a specific behavior or mechanism of the distributed system and abstract the rest of the system. Their fundamental advantage is their independence of the execution platform. For example, Bricks (Takefusa et al. 1999) was proposed for studies and comparisons of scheduling algorithms and frameworks. Researchers can specify network topologies, server architectures, communication models and scheduling framework components to study multi-client, multi-server Grid scenarios. Some Bricks components are replaceable by real software, allowing validation of external sofware. SimGrid (Casanova, Legrand, and Marchal 2003) is used to study single-client multi-server scheduling in the context of complex, distributed, dynamic, and heterogeneous environments. SimGrid is based on event-driven simulation, providing a set of abstractions and functionalities to build a simulator corresponding to the applications and infrastructures. Resources latency and service rate may be set as constants or evolve according to traces. The topology is fully configurable. GangSim (Dumitrescu and Foster 2005) considers a context where hundreds of institutions and thousands of individuals collectively use tens or hundreds of thousands of computers and the associated storage systems. It models usage policies at the site and Virtual Organization (VO) levels and can combine simulated components with instances of a VO Ganglia Monitoring toolkit running on real resources.

Surprisingly, very few studies have provided validation for these simulators. The validation of Bricks was performed by incorporating NWS (Network Weather Service) in Bricks and comparing the NWS results measured on a real Grid with the ones obtained on a Grid simulated by Bricks. SimGrid validation consisted in comparing the simulator results with the ones obtained analytically on a mathematically tractable problem.

In some situations, complex behaviors and interactions of the distributed system nodes cannot be simulated, because of the difficulty of capturing and extracting the factors influencing the distributed systems. Emulators can address this limitation by executing the actual software part of the distributed system, in its whole complexity. Emulators are generally run on rather ideal infrastructures (i.e. controlled clusters). MicroGrid (Liu, Xia, and Chien 2004) allows researchers to run Grid applications on virtual Grid resources. Resource virtualization is done by intercepting all direct use of resources. The emulation coordination essentially controls the simulation rate, which is determined by the virtualization ratio for all resources. The emulation time base is controlled by a virtualization library returning adjusted times to the system routines. Accurate processor virtualization relies on specific schedulers and the network virtualization (Liu, Xia, and Chien 2004) uses the MaSSF system for a scalable online network simulation.

The authors of MicroGrid have conducted a thorough validation (Liu, Xia, and Chien 2004). The internal timing of MicroGrid was validated using the AutoPilot system. The capacity of the emulator to enforce memory limitation and to maintain the processing model under CPU and I/O competition was validated using microbenchmark. Emulation results were compared with experimental ones on real platforms for the NAS benchmark, in order to validate the full emulation engine. Validation with real applications compared the execution times of CACTUS problem solving environment, Jacobi, ScaLAPACK, Fish, Game of life, and Fasta on real platforms with the ones obtained by MicroGrid. Emulab (White et al. 2002) is another emulator, originally designed for network emulation. It provides advanced controlling mechanisms for the user, allowing the rebooting of nodes in specific OS configurations and the control of the network topology.

Because emulators use the real software, they cannot scale as well as simulators. Furthermore, there is still a gap between emulators and the reality: even traffic and fault injection techniques, generally based on traces or synthetic generators cannot capture all the dynamic, variety and complexity of real-life conditions. Real-life experimental platforms solve this problem by running the real software on realistic hardware. DAS2 (http://www.cs.vu.nl/das2/) is basically an idealized Grid, all sites being connected on the Internet. Experiments are run on top of a Grid middleware managing the classical security and runtime interface issues related to Grid platforms. The nodes are voluntarily homogeneous, providing a much simpler management and helping a better environment for performance comparison (speed up of parallel applications) and understanding. PlanetLab (Chun et al. 2003) is another real-life experimental platform, connecting real machines through the Internet, at the planet scale. Some production Grids (TeraGrid, eScience, DataGrid) have also been used as experimental platforms, before being opened to actual users or during dedicated time slots.

Two major limitations of real-life platforms as experimentation tools are 1) their low software reconfiguration capability and 2) the lack of deep control and monitoring mechanisms for the users. The next section highlights how Grid'5000 addresses these limitations.

3 Designing Grid'5000

The design of Grid'5000 derives from the combination of 1) the limitations observed in simulators, emulators and real platforms and 2) an investigation into the research topics conducted by the Grid community. These two elements led to the proposal for a large scale experimental tool, with deep reconfiguration capability, a controlled level of heterogeneity and a strong control and monitoring infrastructure.

3.1 Experiment Diversity

During the preparation of the project (2003), we asked researchers in Grid computing which experiments they were willing to conduct on a large scale real-life experimental platform. The members of 10 teams in France, involved in different aspects of Grid computing and well connected to the international Grid community, proposed a set of about 100 experiments. It was surprising to discover that almost all teams required different infrastructure settings for their experiments. The experiment diversity nearly covered all layers of the software stack used in Grid computing: networking protocols (improving point to point and multipoints protocols in the Grid context, etc.); operating systems mechanisms (virtual machines, single system image, etc.); Grid middleware; application runtimes (object oriented, desktop oriented, etc.); applications (life science, physics, engineering, etc.); problem solving environments. Research in these layers concerns scalability (up to thousands of CPUs), performance, fault tolerance, QoS, and security.

3.2 Deep Reconfiguration

For researchers involved in network protocols, OS and Grid middleware research, the software setting for their experiments often requires specific OS. Some researchers need Linux, while others are interested in Solaris10 or Windows. For networking research, FreeBSD is preferred because network emulators such as Dummynet and Modelnet run only on this operating system. Some researchers also need to test and improve protocol performance (for example changing the size of the TCP window or testing alternative protocols). Some research on virtual machines, process checkpointing and migration need the installation of specific OS versions or OS patches that may not be compatible with each other. Even for experiments over the OS layers, researchers have some preferences: for example some prefer Linux kernel 2.4 or 2.6 because their schedulers differ. Researchers' needs are quite different in Grid middleware: some require Globus (in different versions: 3.2, 4, DataGrid version) while others need Unicore, Desktop Grid or P2P middleware. Some other researchers need to make experiments without any Grid middleware and test applications and mechanisms in a multi-site, multi-cluster environment before evaluating the Middleware overhead. According to this inquiry on researchers' needs, Grid'5000 should provide a deep reconfiguration mechanism allowing researchers to deploy, install, boot and run their specific software images, possibly including all the layers of the software stack. In a typical experiment sequence, a researcher reserves a partition of Grid'5000, deploys its software image, reboots all the machines of the partition, runs the experiment, collects results and relieves

the machines. This reconfiguration capability allows all researchers to run their experiments in the software environment exactly corresponding to their needs.

3.3 A Two-Level Security Approach

Because researchers must be able to boot and run their specific software stack on Grid'5000 sites and machines, we cannot make any assumption on the correct configuration of the security mechanisms. As a consequence, we should consider that Grid'5000 machines are not protected. Two other constraints increase the security issue complexity: 1) all the sites hosting the machines are connected through the Internet and 2) basically inter-site communication should not suffer any platform security restriction and overhead during experiments. From this set of constraints, we decided to use a two-level security design with the following rules: a) Grid'5000 sites are not directly connected to the Internet and b) all communication packets fly without limitation between Grid'5000 sites. The first rule ensures that Grid'5000 will resist hacker attacks and will not be used as basis of attacks (i.e. massive DoS or other more restricted attacks).

These design rules led to building a large scale confined cluster of clusters. Users connect to Grid'5000 from the lab where the machines are hosted. Rigorous authentication and authorization check is done first to enter the lab and then to log in Grid'5000 nodes from the lab. In order to participate in multiplatform experiments, it is possible for Grid'5000 sites to open restricted routes through the Internet to external clusters (called satellite sites).

3.4 Two Thirds as Homogeneous Nodes

Performance evaluation in Grid is a complex issue. Speedup evaluation is hard to evaluate with heterogeneous hardware. In addition, the hardware diversity increases the complexity of the deployment, reboot and control subsystem. Moreover, multiplying the hardware configurations directly leads to an increase in the everyday management and maintenance cost. Considering these three parameters, we decided that 2/3 of the total machines should be homogeneous. However, Grid are heterogeneous by nature and this is an important dimension in the experiment diversity. This is the reason why we chose to keep 1/3 as heterogeneous machines.

3.5 Precise Control and Measurement

Grid'5000 is used for Grid software evaluation and making fair comparisons of alternative algorithms, software, protocols, etc. This implies two elements: first, users should be able to steer their experiments in a reproducible way and second, they should be able to access probes providing precise measurements during the experiments. The reproducibility of experiment steering includes the capability to 1) reserve the same set of nodes, 2) deploy and run the same piece of software on the same nodes, 3) synchronize the experiment execution on all the involved machines, 4) if needed, repeat sequences of operations in a timely and synchronous way, 5) inject the same experimental conditions (synthetic or trace based: fault injection, packet loss, latency increase, bandwidth reduction). As described in the next section, Grid'5000 software set provides a reservation tool (OAR, see Georgiou et al. 2005), a deployment tool (Kadeploy2, see Georgiou et al. 2006) and several experimental condition injectors. Precise and extensive measurement is a fundamental aspect of experimental evaluation on real-life platforms.

Global observation of the network (from its edges) and local observation of processor, memory or disk is difficult at the hardware level and since the users may use their own software configuration, there is no way to provide a built-in and trustworthy monitoring system for CPU, memory and disc. Hence, it is the responsibility of the users to properly install, configure and manage the software observation tools they need for their experiments.

4 Grid'5000 Architecture

The Grid'5000 architecture implements the principles described in the previous section. Based on the researchers requirements, the scalability needs and the number of researchers, we decided to build a platform of 5000 CPUs distributed over 9 sites in France. Figure 2 presents

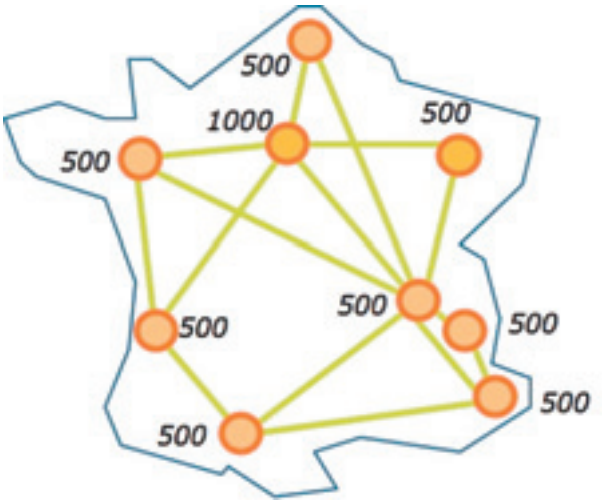


Fig. 2 Overview of Grid'5000.

an overview of Grid'5000. Every site hosts a cluster and all sites are connected by high speed network (a novel network architecture is being deployed, connecting the sites with 10 Gbps links).

Numbers in Figure 2 give the target number of CPUs for every cluster. Two-thirds of the nodes are dual CPU 1U racks equipped with 2 AMD Opteron processors running at 2 GHz, 2 GB of memory and two 1Gbps Ethernet Adapters. Clusters are also equipped with high speed networks (Myrinet, Infiniband, etc.). In the rest of this section we present the key architectural elements of Grid'5000.

4.1 A Confined System

As discussed earlier, the Grid'5000 architecture should provide an isolated domain where communication is allowed without restriction between sites and is not possible directly with the outside world. Mechanisms based on state-of-the-art technology such as public key infrastructures and X509 certificates, produced by the Grid community to secure all resources accessed are not suitable for the Grid'5000. The GSI high level security approach imposes a heavy overhead and impacts on the performance, biasing the results of studies not directly related to security.

A private dedicated network (PN) or a virtual private network (VPN) are, then, the only solutions to compose a secure grid backbone and to build such a confined infrastructure. In Grid'5000, we chose to interconnect the sites with a combination of DiffServ and MPLS technology (Multiprotocol label switching) provided by RENATER (our service provider). MPLS is an efficient way to build secure virtual private networks. As the packet encapsulation is done at very low level by very high performance routers, the overhead is negligible and has no impact on the end-to-end performance. One difference between classical Internet and MPLS, is that MPLS fixes the routing of Grid'5000 datagrams and flows. We consider that the static routing constraint is reasonable for such a testbed. Concerning the background traffic, and the meaningfulness of our end-to-end measures, we chose to let the researcher load the network with artificially generated traffic he can monitor rather than letting him deal with unknown Internet traffic. It allows the calibration of tools, the debugging of protocols and the investigation of alternative traffic control strategies and different types of traffic models. It appears Grid'5000 is complementary to PlanetLab as our instrument enables fine tuning and good understanding of basic phenomenon in the absence of extra noise. PlanetLab offers a more realistic view of the present Internet behavior, but cannot capture behaviors at the limit of the resource capacities and potentially the future Internet behavior.

Many VPN implementation solutions are available but they do not provide security and QoS guarantees simultaneously. For security, network layer VPNs may use tunneling or network layer encryption (layer 3 VPN). A link layer, VPNs such as MPLS are directly provided by network service providers (layer 2-3 VPN). The advantage of the MPLS VPN over IP VPN (Ipsec) is performance. As Grid'5000 sites are connected to the same NREN (National Research and Education Network), the multi-domain issue of the MPLS technology is avoided here. For performance guarantee, a combination of DiffServ and MPLS will be configured for Grid'5000 links. The Premium service will be used for delay and bandwidth guarantees required for reproducible experimental conditions and performance measurements. This MPLS-based Grid architecture allows the creation of a trust context that even enables to experiment with new security solutions for IP VPN-based Grids. Figure 3 presents the resulting communication architecture.

Using MPLS in Grid architecture is not an isolated choice. Recently, a Grid VPN research group was born within the GGF, attesting a real interest in developing and using MPLS, G-MPLS or lower level optical switching technologies for the Grid.

4.2 User View and Data Management

As previously mentioned, communications are done with minimal authentication between Grid'5000 machines. The logical consequence is that a user has a single account across the whole platform. However, each Grid'5000 site manages its own user accounts. Reliability of the authentication system is also critical. A local network outage should not break the authentication process on other sites. These two requirements have been fulfilled by the installation of an LDAP directory. Every site runs an LDAP server containing the same tree: under a common root, a branch is defined for each site. On a given site, the local administrator has read-write access to the branch and can manage its user accounts. The other branches are periodically synchronized from remote servers and are read-only.

From the user's point of view, this design is transparent. Once the account is created, the user can access any of the Grid'5000 sites or services (monitoring tools, Wiki, deployment, etc.). His data, however, are local to every site. They are shared on any given cluster through NFS, but distribution to another remote site is done by the user through classical file transfer tools (rsync, scp, sftp, etc.). Data transfers with the outside of Grid'5000 are restricted to secure tools to prevent identity spoofing and public key authentication is used to prevent brute-force attacks.

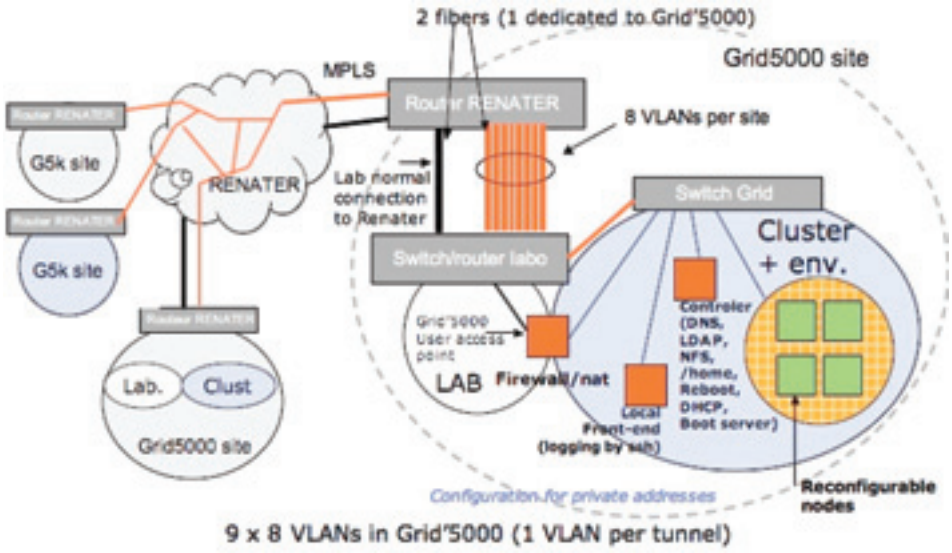


Fig. 3 Communication architecture.

4.3 Experiment Scheduling

Experiment scheduling and resource allocation is managed by a resource management system called OAR (Georgiou et al. 2005) at cluster level and by a simple broker at the grid level. OAR architecture is built from a relational database engine *MySql*. All large-scale operations such as parallel task launching, node probing or monitoring are performed using a specialized parallel launching tool named *Taktuk* (Augerat, Martin, and Stein 2002). OAR provides most of the important features implemented by other batch schedulers such as priority scheduling by queues, advance reservations, backfilling and resource match making.

At grid level, a simple broker allows co-allocating sets of nodes on every selected cluster.

The co-allocation process works as follows: 1) user submits an experiment which needs several sets of nodes on different clusters; 2) in round-robin sequence, the broker submits a reservation to each local batch scheduler. If one reservation is refused, all previously accepted reservations are canceled. When all local reservations are accepted, the user receives an identifier from the broker, allowing the user to retrieve information about the allocated set of nodes.

In Grid'5000, the resource management system is coupled with node reconfiguration operation at different points. First, a specific queue is defined where users can submit experiments requesting node reconfiguration. Second, there is a dynamic control of deployment rights in the prologue script that is executed before starting the experiment. This gives the user the capability of deploying system images on the allocated node partition. Rights are revoked in the epilogue script after the experiment. Third, after the completion of experiments involving node reconfiguration, all nodes are rebooted in a default environment. This default environment provides libraries and middleware for experiments without reconfiguration.

4.4 Node Reconfiguration

Node reconfiguration operation is based on a deployment tool called Kadeploy2 (Georgiou et al. 2006). This tool allows users to deploy their own software environment on a disk partition of selected nodes. As previously mentioned, the software environment contains all software layers from OS to application level needed by users for their experiments.

The architecture of Kadeploy2 is also designed around a database and a set of specialized operating components.

The database is used to manage different aspects of the node configuration (disk partition schemes, environment deployed on every partition), user rights to deploy on nodes, environment description (kernel, initrd, custom kernel parameters, desired filesystem for environment, associated postinstallation) and logging of deployment operations.

Several deployment procedures are available, depending mainly on OS type and filesystem specificity. We only sketch the usual deployment procedure. First, when a user initiates a deploy operation, he provides an environment name allowing the retrieval of associated information from the database. The user provides this information at environment registration. Deployment begins by rebooting all nodes on a minimal system through a network booting sequence. This system prepares the target disk for deployment (disk partitionning, partition formatting and mounting). The next step in the deployment is the environment broadcast which uses a pipelined transfer between nodes with on-the-fly image decompression. At this point, some adjustments must be done on the broadcasted environment in order to be compliant with node and site policies (mounting tables, keys for authentication, information for specific services that cannot support auto-configuration). The last deployment step consists in rebooting the nodes on the deployed system from a network loaded bootloader.

5 Deployment System Evaluation

In this section, we present the evaluation of the deployment and reboot system of Grid'5000. Evaluation of other parts of Grid'5000 will be presented in future papers. The deployment and reboot system is certainly the most important mechanism of Grid'5000, enabling a rapid turnaround of experiments on the platform. Typical deployment and reboot mechanisms for clusters cannot be coupled to a batch scheduler. Moreover, they are not designed to concurrently install different systems on separate cluster partitions. Our objective is to provide a reconfiguration time (boot-to-boot: B2B) lower than 10 minutes for the 5000 CPUs of the platform. This means: 1) deploying the software image on all the nodes of every site (a site may contain up to 500 nodes); 2) issuing the reboot order on all Grid'5000 nodes; and 3) the reboot of all nodes from the deployed software image. As previously mentioned, Kadeploy2 uses more steps, booting a light kernel to prepare the user partition to boot from for the experiment.

The B2B time depends not only on the performance of Kadeploy2 but also on the OS to be booted (OS have different configurations and run different sets of services). Figure 4 presents the B2B time according to the number of nodes, in a single site, for a simple kernel without service, on a cluster of 200 nodes.

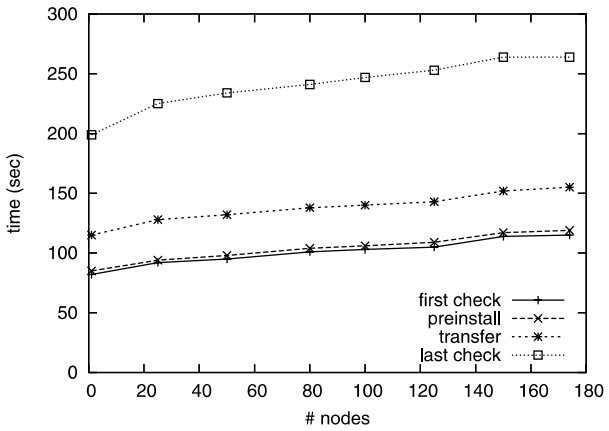


Fig. 4 Time (in seconds) to deploy and boot a new OS on a cluster with Kadeploy2.

The figure presents the completion time of every step included in the B2B time (as a cumulated graph): 1) the time to boot the preparation OS launching a light kernel (first check); 2) the time to prepare the disk partitions before the installation of the user environment (preinstall); 3) the time to transfer the user environment archive (transfer); and 4) the time to boot the user OS (lastcheck). First, the figure shows that the boot time depends on the number of nodes. This is because the boot time is different for all machines and we consider only the slowest one. In contrast, the disk preparation and environment transfer times increase negligibly with the number of nodes. The time to reboot the 2 OS largely dominates the environment transfer time. Altogether, the figure clearly shows a B2B time evolving linearly with the number of nodes following an affine function that could be evaluated as $B2B_{time} = 200 \text{ secs} + (0.33 \times X)$, X being the number of nodes.

Figure 5 presents the time diagram of a deployment and reboot phase involving 2 Grid'5000 sites for a total of 260 nodes (180 nodes in site 1 and 80 nodes in site 2). The vertical axis corresponds to the number of nodes in deployment and reboot states. At $t = 0$ s, all the nodes are running an OS. At $t = 30$ s, a deployment sequence is issued. At $t = 50$ s, all nodes are rebooting the deployment kernel. At $t = 160$ s all nodes have rebooted and are preparing the user partition. The clusters start the second reboot at $t = 200$ s for site 2 and $t = 340$ s for site 1. Site 2 nodes are rebooted with the user OS at $t = 320$ s. All nodes are rebooted with the user OS (including Site 1) at $t = 450$ s. At $t = 800$ s, the user experiment is completed and a reboot order is issued making all nodes reboot to default environment. This figure demonstrates that the current B2B time at the Grid level (450 seconds) is well

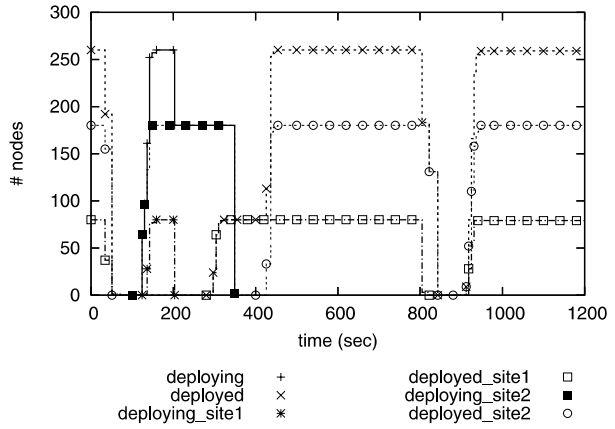


Fig. 5 Time diagram for the deployment and reboot of a user environment on 2 sites.

below the 10 minute mark. The deployment and reboot system is still in Alpha version. It is not tuned and there are many optimization opportunities (Georgiou et al. 2006).

6 Grid'5000 Configuration Examples

The main objective of the Grid'5000 set of software is to ease the deployment, execution and result collection of large scale Grid experiments. In this section, we present 5 examples for Grid'5000 reconfiguration for experiments in networking protocols, Grid middleware infrastructures, and GridRPC environment.

6.1 Testing Recent P2P Protocols in Grid Context

BitTorrent is a popular file distribution system outperforming FTP performance when delivering large and highly demanded files. The key idea of BitTorrent is the cooperation of the downloaders of the same file by uploading chunks of the file to each other. As such, BitTorrent is a nice broadcast protocol for large files in data and computational Grids. BitTorrent uses TCP as the transport protocol.

In this section, we describe how we can deploy, run and collect experiment results, when performing simple BitTorrent performance evaluation for a variation of TCP protocol, on homogeneous nodes of Grid'5000. The modification of the TCP stack involves the compilation and deployment of a specific OS kernel. The experiment requires 9 steps: Step 1) BitTorrent code is instrumented to log reception and emission events (type of communication, sender identifier, receiver identifier, time and chunk identifier). BitTorrent has been instrumented to replay the logged sequence of events. Step 2) The software image is prepared (installing specific libraries and software – Python for BitTorrent), based on a minimal image certified to work on the experimental nodes. The kernel is patched and compiled with alternative TCP versions. The local root file system is then archived and registered on the deploying software database on all sites. Step 3) Nodes are reserved possibly from the same selection file, using OAR. Step 4) The archived file system image is deployed on a user-specified partition of all nodes, using Kadeploy2. Step 5) Kadeploy2 reboots all the reserved nodes and checks that the machine is responding to ping and ssh. 6) The BitTorrent file to be broadcasted, is stored on the user home directory where the BitTorrent master node (the seeder) will run. The list of nodes provided by OAR is stored on the BitTorrent master node. 7) Node clocks are synchronized using NTPdate. 8) A distributed launcher program controls the start of the experiment script on all the nodes. The BitTorrent tracker is started first, then the Torrent file created is registered in the tracker, then the seeder is started on the master and finally, the clients (leechers) are started on all the other nodes. The BitTorrent events are recorded locally on all the nodes. 9) All log files are collected and stored in the user home directory of the user site gateway. Reserved nodes are released.

6.2 Deploying a Globus Toolkit

Globus is an open source grid middleware toolkit used for building grid systems and applications. This part describes how we can map a Globus (Toolkit 2) virtual grid on Grid'5000, deploy Globus, and run experiments. The topology we chose for our virtual Globus grid was to have one Globus installation on each Grid'5000 site. We consider each site to be a separate cluster that provides services through the Globus Toolkit. Since we are emulating a grid, each cluster manages its own user accounts (i.e. no grid-wide user directory). Job execution on clusters is managed by a batch job scheduler (e.g. OAR, PBS). Each cluster manages user accounts and job scheduling with their software of choice, as we only need homogeneity inside clusters. Each site runs a certification authority (CA) that delivers user certificates for their users, as well as host certificates. We pick a front node on each site, and install Globus services on this front node. These services accept requests from other sites, authenticate and authorize them, then perform an action (e.g. submit a job) on behalf of the client. Clients authenticate services with a host certificate delivered by the site the services run on. The Gatekeeper maps user certificates to the user accounts of each cluster, and executes them with the local job scheduler. Front nodes also run the MDS (monitoring and discovery system) service, and GSIFTP (data transfer).

Globus toolkit is deployed by creating a system image that contains a Globus installation tailored for the experiment (since we deploy the whole system image, everything can be customized up to the operating system kernel). We create for each site an image for cluster compute nodes with a batch scheduler, and an image for the front node with the Globus Toolkit services (Gatekeeper, MDS, GSIFTP, and certificates). The virtual Globus grid is deployed on Grid'5000 machines using the Kadeploy tools, thereby turning Grid'5000 into a virtual Globus grid as long as the Kadeploy reservation lasts. While Globus users are running their experiments, log files are saved to the local drives of each node. As soon as the experiment is done, Kadeploy reboots the nodes with their default system image, and users can retrieve their log files and process them.

6.3 A Corba Based Grid Running DIET and TLSE

The DIET (Caron and Desprez 2006) middleware infrastructure follows the GridRPC paradigm (Seymour et al. 2004) for client–server computing over the Grid. It is designed as a set of hierarchical components (client, master and local agents, and server daemons). It finds an appropriate server according to the information provided in the client request (problem to be solved, size of the data involved), the performance of the target platform (server load, available memory, communication performance), and the availability of data stored during previous computations. The scheduler is distributed using several hierarchies connected either statically (in a Corba fashion) or dynamically (in a peer-to-peer fashion).

The main goal of the Grid-TLSE project (Dayde et al. 2004) is to design an expert site that provides an easy access to a number of sparse matrix solver packages allowing their comparative analysis on user-submitted problems, as well as on matrices from collections also available on the site. The site provides user assistance in choosing the right solver for its problems and appropriate values for the solver parameters. A computational Grid managed by DIET is used to deal with all the runs related to user requests. Our goal in the Grid'5000 project is twofold. First we want to validate the scalability of our distributed scheduling architecture at a large scale (using thousands of servers and clients) and then to test some deployments of the TLSE architecture for future production use.

In the current availability of Grid'5000 platform, the deployment of DIET with TLSE server works in three phases. The first step consists in sending one OAR request at each site, to reserve a maximum of available nodes. The second phase consists in receiving OAR information to know which nodes are given by reservation. The third phase generates an XML file with the dynamic information as well as names of nodes at each site. These files will be used by GoDIET to deploy DIET. Our main goal during this first experience is to corroborate a theoretical study of the deployment with the hardware capability of Grid'5000 platform (CPU performance, bandwidth, etc.) to design a hierarchy that achieves a good scalability and a good efficiency for DIET. From this XML file, GoDIET deploys agents (or schedulers), servers and services bound to DIET as Corba services (i.e. naming service) along with a distributed log tool designed for the visualization tools (VizDIET, see Bolze, Caron, and Desprez 2006). Figure 6 shows a large deployment of DIET using 574 computing nodes and 9 agents for the scheduling of 45000 requests. The 574 servers are deployed on 8 clusters and 7 sites.

6.4 Process Design, Optimization, Planning and Scheduling

Process systems engineering is concerned with the understanding and development of systematic procedures for the design and operation of chemical process systems, ranging from continuous to batch processes at industrial scale (Ponish et al. 2005). More precisely, the optimal design continuous process consists in selecting simultaneously the unit operations, the topology and the best operating conditions. Several software tools have been developed for solving this type of problems. One of them, AG (1,2), used at the LGC (Laboratoire de Genie Chimique) is a serial fortran code. To give an illustration, the treated problem instances involve problem sizes of between 170 and 210 variables. Half of them are integer, which corresponds to a combinatorial aspect of about 1.e40 and 1.e50 (since 3 values are possible for each integer variable). The problem is identified as an NP-hard problem.

This application is multi-parametric by nature since it uses a stochastic algorithm where the execution is repeated 100 times with different parameters. Each execution requires 4 hours on the previous example giving a total execution time of 400 hours on a single PC.

Gridification of such applications is straightforward. The first step consists in modifying the code in order to be able to schedule the 100 executions over 100 different nodes. The code has then been deployed over 5 clusters of Grid'5000: Lyon, Orsay, Sophia, Bordeaux, and Toulouse. The code does not reference any external library which simplifies greatly the installation process at every site.

Getting 100 nodes over 5 clusters of Grid'5000 is usually not a problem and the elapsed time for simulation is reduced from 400 hours down to 4 hours. There is an obvious benefit: the possibility of solving larger problems. But a major benefit compared with traditional large computer infrastructures is that the time between execution launching (usually through a batch system that may

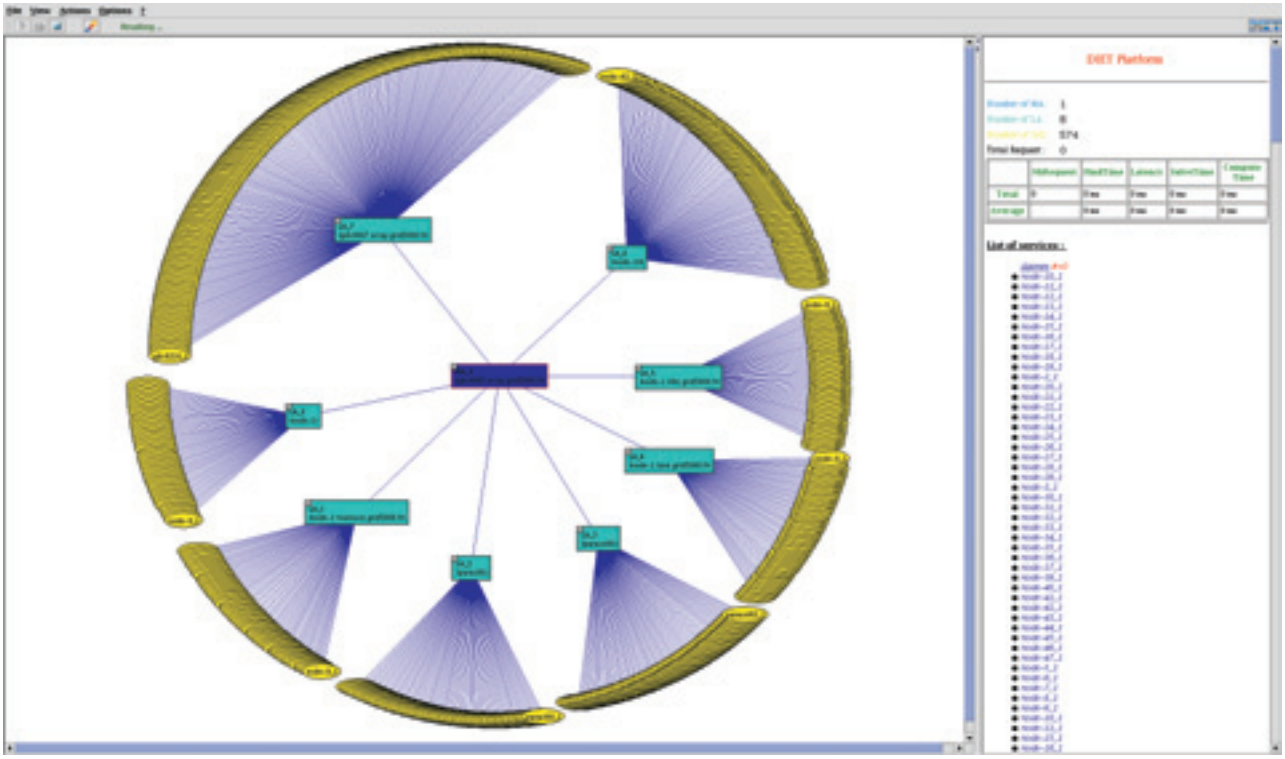


Fig. 6 Large DIET deployment on Grid'5000.

limit the maximal number of processors to a lower number and impose a long wait on a specific queue) and result recovering is drastically reduced, allowing researchers to carry out more simulations.

6.5 The Flow-Shop Challenge on Grid'5000

The Flow-Shop problem consists roughly in finding a schedule of a set of jobs on a set of machines that minimizes the total execution time called the *makespan*. The jobs must be scheduled in the same order on all machines, and each machine cannot be simultaneously assigned to two jobs. The complexity of the problem is very important for large size instances in terms of potential solutions (i.e. schedules). Even with a modern workstation the resolution based on an exhaustive enumeration of all possible combinations would take several years. Therefore, the challenge is to reduce the number of explored solutions using efficient algorithms in order to solve the problem in a reasonable time. Nevertheless, even if these algorithms allow significant reduction in the size of the search space the complexity remains high, and the problem could not be efficiently solved without computational grids.

To solve the problem, a new grid exact method based on the Branch-and-Bound (B&B) algorithm has been proposed by Melab (2005). The method is based on a large scale dispatcher–worker cycle stealing approach. The dispatcher controls the exploration of the search tree generated by the distributed B&B algorithm. It maintains the best solution found so far and a pool of work units and ensures their dynamic allocation to the different workers joining the computational grid. Each work unit represents a set of nodes (or solutions) to be or being explored, and is designated by a small descriptor. Each worker explores its assigned tree nodes using the B&B algorithm and sends back a solution to the dispatcher if it is better than the best known solution so far. To deal with the load bal-

ancing issue, as soon as its local pool of nodes to be explored is empty each worker requests from the dispatcher a work unit. The dispatcher selects a work unit being executed and splits it in two parts. The second part, which is probably not yet explored, is sent to the worker asking for work.

Another issue which is dealt with in the proposed grid exact method is the fault tolerance. This issue arises on a computational grid because of failures of resources (processors, networks, etc.) or their dynamic availability. Within the Grid'5000 context, the dynamic availability is a result of the reservation policy of the grid. Indeed, for long-running applications, a series of reservations is required to resume their execution. The end of each reservation is put in the same category as a failure of the corresponding resources. In the proposed method, a checkpointing-based approach is proposed to deal with the fault-tolerance issue. Each worker periodically requests the dispatcher to update the descriptor of its associated work unit with its current state. The descriptor is based on a special coding of the search tree which allows minimizing of the memory space required by the checkpointing mechanism and the communications involved by the dynamic work distribution.

The proposed method has been implemented using the XtremWeb middleware (Mezmaz, Melab, and Talbi 2006) and RPCs. The second implementation is used to solve the Taillard's Flow-Shop problem instance $Ta056$[1] of scheduling 50 jobs on 20 machines. A near-optimal solution has been found in (Ruiz and Stutzle 2004). However, as it is CPU time consuming such instance has never been optimally solved. The proposed method in Melab (2005) allowed not only an improvement in the best known solution (Ruiz and Stutzle 2004) for the problem instance but also proved the optimality of the provided solution. Indeed, once the supposed optimal solution is found it has to be compared with the remaining solutions being visited to prove that it is really the best.

The experiments were performed on a computational grid including, simultaneously, processors from Grid'5000 and different educational networks of Université de Lille1 (Polytech'Lille, IEEA, IUT "A"). The number of processors averaged approximately 500, and peaked at 1245 machines during one night. The Grid'5000 sites involved in the computation are Bordeaux, Lille, Orsay, Rennes, Sophia-Antipolis and Toulouse. The optimal solution was found with a total wall-clock time of 7 weeks. The experiment was performed a second time starting from the best known near-optimal solution minus 1. The optimal solution was found within 25 days and 46 minutes. The resolution would take 22 years 185 days and 16 hours on a single machine. During the resolution, an average of 328 processors were used and peaked at 1195 processors. The total number of explored nodes was $6{,}5874e+12$, and a small number of them (0. 39%) were explored twice (redundant work). Moreover, 129 958 work allocation operations and 4 094 176 checkpointing opeartions have been performed. Such statistics show that the dynamic load-balancing and checkpointing mechanisms have been heavily requested and performed well. Furthermore, the parallel efficiency is measured as the ratio between the agregated execution time of the workers and the total time of their availability. The parallel efficiency observed in this experience is 97%, so the load balancing approach is very efficient. Finally, the average CPU time consumed by the dispatcher is 1.7%. The approach scales up to 1195 processors without any problem.

7 Conclusion

Grid'5000 belongs to a novel category of research tools for Grid research: a large scale distributed platform that can be easily controlled, reconfigured, and monitored. We have presented the motivation behind design and architecture of this platform. The main difference between Grid'5000 and previous real-life experimental platforms is its degree of reconfigurability, allowing researchers to deploy and install the exact software environment they need for each experiment. This capability raises a security difficulty, solved in Grid'5000 by establishing a virtual domain spanning over several sites, rigorously controlling the communications at the domain boundaries and relaxing restrictions for intra-domain communications. We have described some configuration examples, illustrating the variety of experiments that can benefit from Grid'5000. We also presented the performance of the reconfiguration system which provides a "boot-to-boot" time of less than 10 minutes on the full platform.

Ongoing work focuses on several areas: 1) ease software image construction for the users; 2) provide automatic validation of software images; 3) support and coordinate experiments; and 4) tune and validate network performance.

For more information about the Grid'5000 project, please contact the corresponding author, Franck Cappello (fci@lri.fr), who is responsible for this project.

Acknowledgments

We would like to thank the French Ministry of research and the ACI Grid and ACI Data Mass incentives, especially Thierry Priol (Director of the ACI GRID) and Brigitte Plateau (Head of the Scientific Committee of the ACI Grid), and Dany Vandromme (Director of RENATER) for their support. We also thank INRIA, CNRS, regional councils of Aquitaine, Bretagne, Ile de France and Provence Alpe Côte d'Azur, Alpes Maritimes General Council and the following Universities: University of Paris Sud,

Orsay, University Joseph Fourier, Grenoble, University of Nice-Sophia Antipolis, University of Rennes 1, Institut National Polytechnique de Toulouse/INSA/FERIA/ Universite Paul Sabatier, Toulouse, University Bordeaux 1, University Lille 1/GENOPOLE, Ecole Normale Superieure de Lyon. We also thank MYRICOM.

Author Biographies

Raphaël Bolze is a Ph.D. student at Ecole Normale Superieure de Lyon. He received his Masters in computer science in 2003 from the Institut de Recherche en Informatiques de Nantes and also a Masters degree in engineering from l'Ecole Polytechniques de l'Universite de Nantes. His research interests focus on workflow scheduling over grid environment.

Franck Cappello holds a Research Director position at INRIA and leads the Grand-Large project at INRIA. He has initiated the XtremWeb (Desktop Grid) and MPICH-V (Fault tolerant MPI) projects. He is currently the director of the Grid'5000 project, designing, building and running a large scale Grid experimental platform. He has authored more than 60 papers in the domains of high performance programming, desktop grids, grids and fault tolerant MPI. He has contributed to more than 30 Program Committees. He is editorial board member of the International Journal on GRID Computing and steering committee member of IEEE HPDC and IEEE/ACM CCGRID. He is the general chair of IEEE HPDC'2006.

Eddy Caron is an assistant professor at Ecole Normale Superieure de Lyon and holds a position with the LIP laboratory (ENS Lyon, France). He is a member of GRAAL project and technical manager for the DIET software package. He received his Ph.D. in computer science from University de Picardie Jules Verne in 2000. His research interests include parallel libraries for scientific computing on parallel distributed memory machines, problem solving environments, and grid computing.

Michel J. Daydé received his Ph.D. from Institut National Polytechnique de Toulouse (France) in 1986 in computer science. From 1987 to 1995, he was a postdoctorate fellow then visiting a Senior Scientist in the Parallel Algorithms Group at CERFACS. From 1988, he has been Professor at Ecole Nationale Superieure d'Electrotechnique, d'Electronique, d'Informatique, d'Hydraulique et des Télécommunications (ENSEEIHT) at Institut National Polytechnique de Toulouse. Since 1996, he has been Research Director in the Groupe Algorithmes Paralleles et Optimisation at Institut de Recherche en Informatique de Toulouse (IRIT). He is Head of the ENSEEIHT Site of IRIT and Vice-Head of IRIT. His current research interests are in grid computing, parallel computing and computational kernels in linear algebra and large scale nonlinear optimization. He is the coordinator of the GRID-TLSE Project and scientific coordinator of the Toulouse/Midi-Pyrenees Site of GRID'5000.

Frédéric Desprez is a director of research at INRIA and holds a position at LIP laboratory (ENS Lyon, France). He received is Ph.D. in computer science from the Institut National Polytechnique de Grenoble in 1994 and his M.S. in computer science from the ENS Lyon in 1990. His research interests include parallel libraries for scientific computing on parallel distributed memory machines, problem solving environments, and grid computing.

Emmanuel Jeannot is currently full-time researcher at INRIA (Institut National de Recherche en Informatique et en Automatique) and is doing its research at the LORIA laboratory. From September 1999 to September 2005 he was associate professor at the Université Henry Poincare, Nancy 1. He got his Ph.D. and Master degree of computer science (respectively in 1996 and 1999) both from Ecole Normale Superieure de Lyon. His main research interests are scheduling for heterogeneous environments and grids, data redistribution, grid computing software, adaptive online compression and programming models. He is currently visiting the ICL Laboratory of the University of Tennessee.

Yvon Jégou is a full time INRIA reasercher in the PARIS project of INRIA-Rennes (IRISA). His research activities are centered on architecture, operating systems and compilation techniques for parallel and distributed computing. His current work is focused on the development of a DSM for the implementation of runtime systems on large clusters and for the management of data repositories on the Grid. In the recent past, he participated to the IST POP European project on the implementation of an OpenMP system for clusters using distributed shared memories (DSM). He is currently involved in the XtreemOS European project. The objective of XtreemOS is the development of a Grid operating system with native support for virtual organizations. He is the leader of the Grid'5000 team at INRIA-Rennes.

Stephane Lanteri is a researcher at INRIA Sophia Antipolis in a scientific computing team. His current activities are concerned with the design of unstructured mesh based numerical methods for the discretization of PDE systems modeling wave propagation phenomena, domain decomposition and multilevel algorithms and high performance parallel and distributed computing. He is the scientic coordinator of the Grid5000@Sophia project which defines the contributions of INRIA Sophia Antipolis to the Grid'5000 project.

Julien Leduc is the contractor CNRS Research Engineer, a member of Grid'5000 technical committee and participated in the design of the Grid'5000 grid services architecture. He is the technical manager of the reconfiguration feature of Grid'5000: Kadeploy designer and main developer. Previously, he worked on the Clic clustering distribution, and system administration of several clusters in Grenoble.

Nordine Melab received his Master's, Ph.D. and HDR degrees in computer science, from the Laboratoire d'Informatique Fondamentale de Lille (LIFL, Universite de Lille1). He is an Associate Professor at Polytech'Lille and a member of the OPAC team at LIFL. He is involved in the DOLPHIN project of INRIA Futurs. He is particularly a member of the Steering Committee of the Grid'5000 French Nation-Wide project. His major research interests include parallel and grid computing, combinatorial optimization algorithms and applications and software frameworks.

Dr. Pascale Vicat-Blanc Primet, graduated in Computer Science, is senior researcher (Directrice de Recherche) at INRIA. Her research interests include Distributed and Real-Time Systems, High Performance Grid and Cluster Networking, Active Networks, Internet protocols (TCP/IP), Network Quality of Service. She is leading the RESO team, labelled RESO project of the Institut National de la Recherche en Informatique et Automatique (INRIA) at LIP laboratory in LYON (France). Since 2000, she has been very active in the international and national Grid community. Co-chair of the DataTransport Research Group in the Global Grid Forum, she has co-edited several Grid Networking and Transport protocol GGF documents. She is general co-chair of the International GRIDNETS conference and PFLDNET workshop, member of international conferences steering or program committees and reviewer for international conference and journal in Grids and Networking. She has published her work in more than 60 papers in Grid and Networking journals or conferences. She is member of the steering committee of the French ACI MD DataGRID Explorer (GdX) project, of the ACI Grid Grid5000 project, ANR IGTMD and EU Strep EC-GIN project.

Raymond Namyst received his Ph.D. in computer science from Lille in 1997. He held the position of assistant professor in the Computer Science Departement of the Ecole Normale Superieure of Lyon (1997–2002). In 2002, he joined the Computer Science Laboratory of Bordeaux (LaBRI) where he holds a Professor position. He is the head of the Runtime INRIA research project, devoted to the design of high performance runtime systems for parallel architectures. His main research interests are in parallel computing, thread scheduling on multiprocessor architectures, communications over high speed networks and communications within Grids. He has played a major role in the development of the PM2 software suite. He has written numerous papers about the design of efficient runtime systems. He also serves as the chair of the Computer Science Teaching Department of the University of Bordeaux.

Pascale Vicat-Blanc Primet received a Ph.D. degree in computer science from INSA Lyon, France in 1988. Based at École Normale Supérieure de Lyon (ENS-Lyon), she is currently "directrice de recherche" at INRIA and leads the RESO team-project. This team is specialized in communication protocols and software optimization for high-speed networks. Pascale's research interests include high-performance Grid and cluster networking, active networks, TCP, QoS, bandwidth sharing and security. She was a co-chair of the GGF Data Transport Research Group. She is member of the Steering Committee the Gridnets and Pfldnet conferences. Member of the GRID'5000 project steering committee, she is co-charing its Lyon's site. She has published about hundred papers in distributed computing and networking journals and conferences.

Benjamin Quetier is a Ph.D. student of Franck Cappello (INRIA) and works in the Grand-Large project and in the European project CoreGrid. He is also involved in the Grid'5000 project working on virtualization. The goal of his thesis is to build a large scale emulator platform (more than 100 K nodes) over Grid'5000. The first part of his thesis was a comparison of the diverse virtualization tools such as Xen or VMware. He works on the comparison of applications on a real live platform and on a emulated one.

Olivier Richard is an associate professor at the IDIMAG laboratory. He graduated from Paris XI University with a Ph.D. in computer science in 1999. His researh interests are focused on system architecure for high performance computing and large distributed system (cluster, Grid and P2P). His is co-leader of OAR and Kadeploy software projects.

El-Ghazali Talbi received his Master's and Ph.D. degrees in computer science, both from the Institut National Polytechnique de Grenoble. He is presently Professor in computer science at Polytech'Lille (Universite de Lille1), and researcher in Laboratoire d'Informatique Fondamentale de Lille. He is the leader of OPAC team at LIFL, the DOLPHIN project at INRIA Futurs and the platform of bioinformatics of Lille (Genopole de Lille). He took part to several CEC Esprit and national research projects. His current research interests are mainly parallel and grid computing, combinatorial optimization algorithms and applications and software frameworks.

Iréa Touche took part in the project e-toile, which was the first major French high performance data transfer grid project, when she studied to obtain her engineering degree in applied mathematics and scientific calculations. She was in charge of a part of the cluster's configuration of the CEA (Commissariat à l'energie atomique) and also had to deploy a parallel application of molecular dynamics called CHARMM. After that, she worked for 6 months at IRIT (Institut de Recherche en Informatique de Toulouse), on the Grid'5000 project. She had to "gridify" four applications used by different laboratories of INP Toulouse (Institut National Polytechnique). To do this she studied the application's features, and deployed them on one or more clusters of the grid. For the multi-parametric applications, she used several sites in order to be able to easily carry out a great number of executions. For the parallel ones, she has made scalability tests. She is currently working at the LGC (Laboratoire de Génie Chimique), where she helps researchers to optimize their scientific codes.

Note

1 http://ina2.eivd.ch/Collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html

References

Augerat, P., Martin, C., and Stein, B. 2002. Scalable monitoring and configuration tools for grids and clusters. *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*. IEEE Computer Society.

Bolze, R., Caron, E., and Desprez, F. 2006. A monitoring and visualization tool and its application for a network enabled server platform. In LNCS, editor, *Parallel and Distributed Computing Workshop of ICCSA 2006*, 8–11 May, Glasgow, UK.

Caron, E. and Desprez, F. 2006. DIET: A scalable toolbox to build network enabled servers on the Grid. *International Journal of High Performance Computing Applications*, 20(2):335–352.

Casanova, H., Legrand, A., and Marchal, L. 2003. Scheduling distributed applications: the simgrid simulation framework. *Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, Tokyo, Japan.

Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., and Bowman, M. 2003. PlanetLab: An overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12.

Daydé, M., Giraud, L., Hernandez, M., L'Excellent, J.-Y., Pantel, M., and Puglisi, C. 2004. An overview of the grid-tlse project. *Proceedings of 6th International Meeting VECPAR 04*, June, Valencia, Spain.

Dumitrescu, C. and Foster, I. 2005. Gangsim: A simulator for grid scheduling studies. *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05)*, May, Cardiff, UK.

Georgiou, Y., Leduc, J., Videau, B., Peyrard, J., and Richard, O. 2006. A tool for environment deployment in clusters and light grids. *Second Workshop on System Management Tools for Large-Scale Parallel Systems (SMTPS'06)*, April, Rhodes Island, Greece.

Georgiou, Y., Richard, O., Neyron, P., Huard, G., and Martin, C. 2005. A batch scheduler with high level components. *Proceedings of CCGRID'2005*, May, Cardiff, UK. IEEE Computer Society.

Liu, X., Xia, H., and Chien, A. 2004. Validating and scaling the MicroGrid: A scientific instrument for grid dynamics. *The Journal of Grid Computing* 2(2):141–161.

Melab, N. 2005. *Contributions a la resolution de problemes d'optimisation combinatoire sur grilles de calcul*. Ph.D. thesis, November, LIFL, USTL.

Mezmaz, M., Melab, N., and Talbi, E.-G. 2006. A grid hybrid exact approach for solving multi-objective problems. *Proceedings of the 9th IEEE/ACM International Workshop on Nature Inspired Distributed Computing (NIDISC'06 – in conjunction with IPDPS'2006)*, Rhodes Island, Greece.

Ponish, A., Azzaro-Pantel, C., Domenech, S., and Pibouleau, L. 2005. About the relevance of mathematical programming and stochastic optimisation methods: application to the optimal batch plant design problems. *ESCAPE 15*, May 29–June 1.

Ruiz, R. and Stutzle, T. 2004. A simple and effective iterative greedy algorithm for the flowshop scheduling problem. Technical Report, European Journal of Operational Research, in print.

Seymour, K., Lee, C., Desprez, F., Nakada, H., and Tanaka, Y. 2004. The end-user and middleware APIs for GridRPC. *Workshop on Grid Application Programming Interfaces*, in conjunction with GGF12, September, Brussels, Belgium.

Takefusa, A., Matsuoka, S., Aida, K., Nakada, H., and Nagashima, U. 1999. Overview of a performance evaluation system for global computing scheduling algorithms. *HPDC '99: Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing*, Washington, DC, USA. IEEE Computer Society.

White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. 2002. An integrated experimental environment for distributed systems and networks. *OSDI02 Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, pp. 255–270, December, Boston, MA.